COMPUTER METHODS FOR PIPE INTERFERENCE DETECTION IN PLANTS

A Thesis Submitted

In Partial Fulfilment of the

Requirements for the Degree of

MASTER OF TECHNOLOGY
in
ELECTRICAL ENGINEERING

by

BALAKRISHNAN, V.

to the

Department of Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

May 1972

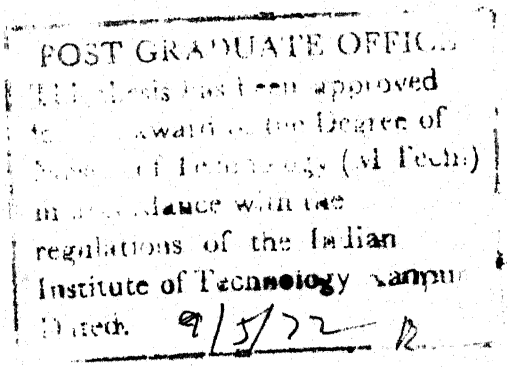
CONTENTS

CERTIFICATE

THIS is to certify that the thesis entitled *"Computer Methods for Pipe Interference Detection in Plants'* is a record of work done by V. Balakrishnan under my supervision, and it has not been submitted elsewhere for a degree.

C.R. Muthukrishnan
Assistant Professor
Electrical Engineering Department
Indian Institute of Technology,

Kanpur
April 28, 1972



With gratitude, I express my thanks to Dr. C.R. Muthukrishnan the project advisor, for his exemplary guidance. He was throughout extremely encouraging and appreciative, and it was a pleasure, and an opportunity to be associated with him.

The meetings with Dr. H.N. Mahabala, though not many, were very enlightening, and some of his suggestions and criticisms helped a lot in refining the package. Besides, as Head of the Computer Centre, he was kind enough to provide me all facilities and complete freedom in using the equipment, and I am particularly grateful to him in this regard.

Dr. V. Srinivas, Assistant Professor in Mechanical Engineering, provided much vital information on the mechanical aspects of pipe layout design. He was extremely helpful in the initial stages of the project.

Mr. V. Suryanarayana of Tata Consulting Engineers (Piping Section) was very kind to explain the techniques adopted in piping inspite of his busy schedule. Mr. S.K. Chaturvedi of the same section was stationed for the project at IIT Kanpur, and gave me good company. I am sure that his association with the project right through its development would stand him well in using the package.

Mr. H.K. Nathani has done a beautiful job of typing, in a really short time.

After all the efforts, it is very much hoped that the package is effectively made use of, as a powerful aid in pipe layout design.

I again thank Dr C.R. Muthukrishnan, Dr. H.N. Mahabala and Dr. Srinivas and the Computer Centre staff for the nice time

PREFACE

This report contains something more than what a normal user would like to know about the inside of the package, but is short of working description of the routines in the package. An engineer who uses a closed package would feel really confident about its performance, only if he has an idea on the logic of the package. In this regard, this report is complete, in that all that an engineer would ever like to know are included. Besides, some of the interesting software problems encountered, and their solutions, are discussed.

Describing three core loads of procedures in detail is perhaps too much for a project report, and is not attempted. The relatively less complex algorithms are left out due to pressure of space. Mention is made of a few procedures which were developed, but could not be implemented, due to want of time.

It may not be as such a very readable report but frequent references to it would be much enlightening to the user. Normally, it would suffice, however, that the user develops a good understanding of the users' manual. The chapters are arranged in the order of relevance to the user, and are each based on the preceding chapters and on the users' manual.

Many of the algorithms dealing with data organization, input/output and pattern description and construction are of use in other applications.

ABSTRACT

The basic principles involved in pipe interference detection are simple but due to the lack of an effective way of representing solids in 3 D space, the conventional methods are very complex, taxing and unreliable. It requires all the efforts of a few skilled engineers, working over a pile of blueprints for an year or more, and still some conflicts come to light only during erection.

By developing an efficient representation of solids in space, and ways of manipulating them in a computer, much of the manual efforts are saved. The first problem is to develop the means of communication - an input language for defining the objects and other constraints and requirements, and a mode of outputting the results of various operations and conflict checks.

The normal plant sizes are too large for completely storing their description and layout in core all the time. An elaborate backup data base is required, alongwith efficient retrieval and updating schemes, which are compatible with the internal representation. Since the backup storage devices (tapes, disk) always pose reliability problems, a complementing set of data protection, error prevention and error recovery schemes become a logical necessity.

A whole set of routines are then required for interpreting the input data and suitably converting them to the corresponding internal representations. In the next phase, the conflicts between the new objects and the existing objects are detected. These conflicts are again in their internal forms, and are to be processed further to generate suitable interpretations for the user.

Due to the amount of data and program area requirements, it is practically impossible to enclose them all in a single core load. An overlay scheme was developed, alongwith an overlay monitor, that controls the loading and execution of appropriate overlay core loads. Further, a bootstrapping scheme minimises the number of cards to be fed in for each run.

The actual plant layout always resides on tapes, and extensively called upon, so that the execution is mostly input/output bound. The availability of two independent data channels, permits extensive overlapping among input/output, and with computation, cutting down execution time by upto 50%. To achieve this, all the algorithms were devised with overlapping in mind, and an efficient buffer switching mechanism has been proposed. The tape operations routine should be able to take care of priority operations.

CHAPTER I

INTRODUCTION

1.1 Motivation

Pipe layout design in a typical plant is a complicated and strenuous affair, consuming upto a few engineer-years. The complications are that the constraints are many and varied, some of them even subjective; and it is strenuous to check each pipe, point by point, for possible conflicts.

Conventional pipe layout proceeds pipe by pipe, in two stages: (i) Deciding a rough path for the pipe and (ii) Extensive verifications for possible conflicts and re-alingnments, if needed. In first deciding a rough pipe path, the engineer has to avoid at least the major structures, objects and prohibited space. The details are not all available in a single drawing, and quite a few drawings will have to be combined mentally before striking at a possible path. He has to keep in mind, besides, the stress considerations, supports and working space considerations and also aesthetic considerations.

It is pratically impossible to arrive at an interference-free pipe path at the first stroke. The second phase is in effect a recurring procedure - a series of rechecking and realingments. There have been cases of

conflicts coming to light during actual erection, calling for last minute changes in layout. This is so because there is no fool proof checking procedure. Ideally, every point of the pipe will have to be checked against so many drawings, to make sure there are no conflicts with structures, bracings, equipment, pathways, prohibited areas, and also other pipes. When the absence of conflict with a particular object is not so obvious from drawing, as is often the case, extensive computations are necessary to make sure that a certain amount of clearance exists. Much time is saved by checking at only significant points, but the selection of these points themselves is partly heuristic and partly arbitrary. The whole approach lacks positiveness, but it reigns, for the lack of any better procedure, that is reasonably fast. It is an extremely skilled procedure, that could tax the resources of the best piping engineers, and yet, it is arbitrary, repetitive and unreliable.

It would hence be very tempting to think of allowing a computer do the job. Basically, the principle involved is extremely simple - that no two objects can occupy the same space. And if there is a systematic way of doing the work, it needs little intelligence; and a computer would be able to do it so many times faster, cheaper and more important, the results would be positive and reliable.

Indeed, the saving achieved is enormous. Pipes can be laid one by one, and we can be sure that no future adjustments would be necessary. The time for layout design can be brought down from a few engineer years to the order of an engineer month; and the erection does not have to wait long. The engineers can forget about the trivial, yet tricky problems, as interference, and concentrate on other considerations, as stress requirements, working space, etc. It would lead to a better utilization of the engineer's skills and knowledge, and also eliminate the monotoneous part of the layout design.

1.2 The State of the Art

Some problems faced in developing pipe interference detection (and pipe layout) programs can be classified as follow

(i) Definition of objects and input format

(ii) Internal representation of objects, and the identification of objects from their internal representations.

(iii) Algorithms for internal manipulations, updating and interference detection

(iv) Creation and maintenance of a data base

(v) Pictorial reconstruction of objects

(vi) Pipe sizing

(vii) Flexibility analysis

Very little relevant work has been reported in any of the fields above. There are a few instances of computer programs being in use for pipe sizing and flexibility analysis. Developed by a few enterprising firms, these are

lent out to others, but the algorithms are not often disclosed. The development of computer programs in these two fields however appear logical, as they involve extensive computations of an established nature, where human computation tends to become too slow, unreliable and tedious. But in the other fields, involving detection of conflicts, and pipe layout design, there is little indication in the literature of any program being developed. There has been much work on the various fields individually, but their possible applications in pipe layout design have not been discussed, and most of them are of little relevance to the problem on hand.

In the fields of definition, internal representation and recognition of three dimensional objects, the progress that has been made is more of academic interest. Interactive languages and grammars have been developed, some of them for special computers (with a high degree of processing parallellism), for defining objects met with in daily life, with particular emphasis on unambiguity and reconstructability. PAX (2) is one such special purpose language, extensively used in image processing as it allows for a wide range of operations and transformations with picture frames. The pictures are stored in a digitized bit pattern representation. Guzman's work (1) on seperating plane bounded objects from an isometric view of a clusture of such objects is of particular interest. He has extensively discussed a simple

way of defining and representing isometric views. Many procedures for defining, composing and encoding of three dimensional objects in terms of basic primitives have also been developed. However, most of these are relevant to us only to the extent of basic ideas suggested and do not meet our requirements of speed and simplicity. Further, our object set being highly standardized, most of the shapes can be internally defined, and this reduces the dependance on the input language for describing shapes.

Digitized bit pattern representation as in PAX (2) has been frequently adopted, when the object size and space are limited. Extending the same representation to a large plant, as in the case of pipe layout design, creates some problems in data organization and handling. No literature has been cited in this direction, as most of the problems handled have a fairly small extent.

R.G. Newell has suggested a method of optimizing the piping cost under fairly simple and idealized constraints (3). Given the nozzle points to be connected, the occupied and prohibited space, and the piping cost at various points, the problem is converted to a "shortest path" finding problem. All the feasible paths, (parallel to the axes only) are considered. All the objects should be approximated to cuboids parallel to the axes. The performance of the method

is however very poor - considering more than fifty items becomes inefficient, the object set is highly limited, and detecting pipe to pipe conflict is not very easy. It is practically impossible to consider even small parts of plants of moderate complexity.

References to a few programs for pipe sizing and flexibility analysis are available in 7,9,10 and 11 without much details on their operation. A rough picture of the present state of computers applications in construction design and erection (including piping) is available in 5,6 and 8. It would be immediately apparent from them that only the analytic part of the various problems have been computerized.

In short, there has been no recorded attempt to computerize the pipe layout problem in its practical form. Detecting the conflicts has remained a hide and seek game needing all the efforts of skilled engineers and a fair bit of providence.

1.3 The Problems

The complexity and the size of a typical plant pose the biggest problems in designing a computer procedure. It is essential that the procedures developed should be able to cope with any plant size and any complexity, at least in a few reasonably large parts. This obviously is beyond the memory capacity of any computer, and an external data base has to be designed on tapes/disks. These devices lack speed, and

the program should be designed to minimise transactions with them for operational speed. The internal representation of the plant and its components is also affected by the above considerations.

Next an efficient way of defining the various shapes is required. It should be fast and simple for the more common shapes, and also versatile enough to define any new shape. Equally important, it should be integral with the internal procedures. Evidently, the language design, the data base design and the procedure design have to go together.

It is as much a necessity to have an efficient feed back system - in the form of readable and relevant messages, instructions and output.

Beyond these basic components, it is desirable to have (1) efficient error detection and intimation mechanism, for errors in data , (2) Recovery procedures for most of the common machine errors, (3) A fair amount of reversibility, to permit more latitude to the user and (4) Mechanisms to assist the engineers in deciding the path of a pipe.

A minimum amount of redundancy would be required for effective reversibility and recovery (after machine errors).

The solutions to the above problems are discussed in length in the following pages.

1.4 Fundamental Principles

The internal representation selected consists of layers of bit patterns, each layer signifying the horizontal section of the plant at a specific level. The horizontal layers are separated by a constant distance = pitch. Within each layer, the individual bits occupy the centre of a square of side = pitch. 'On' bits represent occupied space and 'off' bits represent void. This type of digitized bit patterns at uniformly placed horizontal layers, is termed the standard bit space, and the individual layers are termed standard bit planes. Any bit plane is termed a standard bit plane, only if it occupies exactly the same level as one of the planes in the standard bit space, and within the plane, the positions of the individual bits coincide i.e. there should be no offset in any of the three directions. These bits are incidentally aligned in the vertical plane also, and the vertical planes through the bits are termed standard vertical bit planes.

The above representation can be considered as being composed of horizontal layers of identical cubes (of side = pitch), each cube of space being represented by a bit at the centre of the cube. The safest approach is followed throughout, in that, even if an object grazes one such cube, the whole cube is treated as occupied and the bit turned on.

Evidently, this results in an enlargement of the objects. The extra 'apparent' occupied space is termed 'digital noise' (or simply, noise).

The data base (on tape) consists of one physical record for each standard bit plane of the plant. This record consists of the latest section at the particular level. When an object is added, the pattern to be added for this particular standard bit plane is first composed, and it is superimposed (merging) over the existing pattern on tape. In this process, if bits to be turned on are already on, conflicts are indicated at those points. This procedure is repeated for all those standard bit planes affected by the new object.

The new pattern at each level, obtained by adding the new object to the existing pattern, is stored not in the original place, but in a new place in the tape. (Enough excess space is provided before hand in the data base.) This ensures that the previous version of any standard bit plane is always available - this is made use of extensively in the program.

Due to the digital noise created, it is possible that the conflicts detected are not the real conflicts - they are either too pessimistic, or, only apparant conflicts (or false conflicts). The exact clearances will have to be estimated only by analytic means, if necessary. Also, in

case of joints between, say, a pipe and an equipment, conflict will be detected at the joint. The engineer may suppress these conflicts, to the extent of the joint (termed as conflict suppression) if he desires.

For convenience of handling (as will be explained later), the layers of the plant are handled in groups; A group is a set of contiguous layers, and the size of a group is a constant.

The actual bit patterns reside only on the back-up storage, and are read into and processed in core only during the merging phases. The areas in core where these layers are handled are called the Buffers.

The functions NBIT and POINT are used extensively in the transformations from real space to bit space and vice versa. They are defined as

NBIT (P) = P/PITCH, rounded to next higher integer

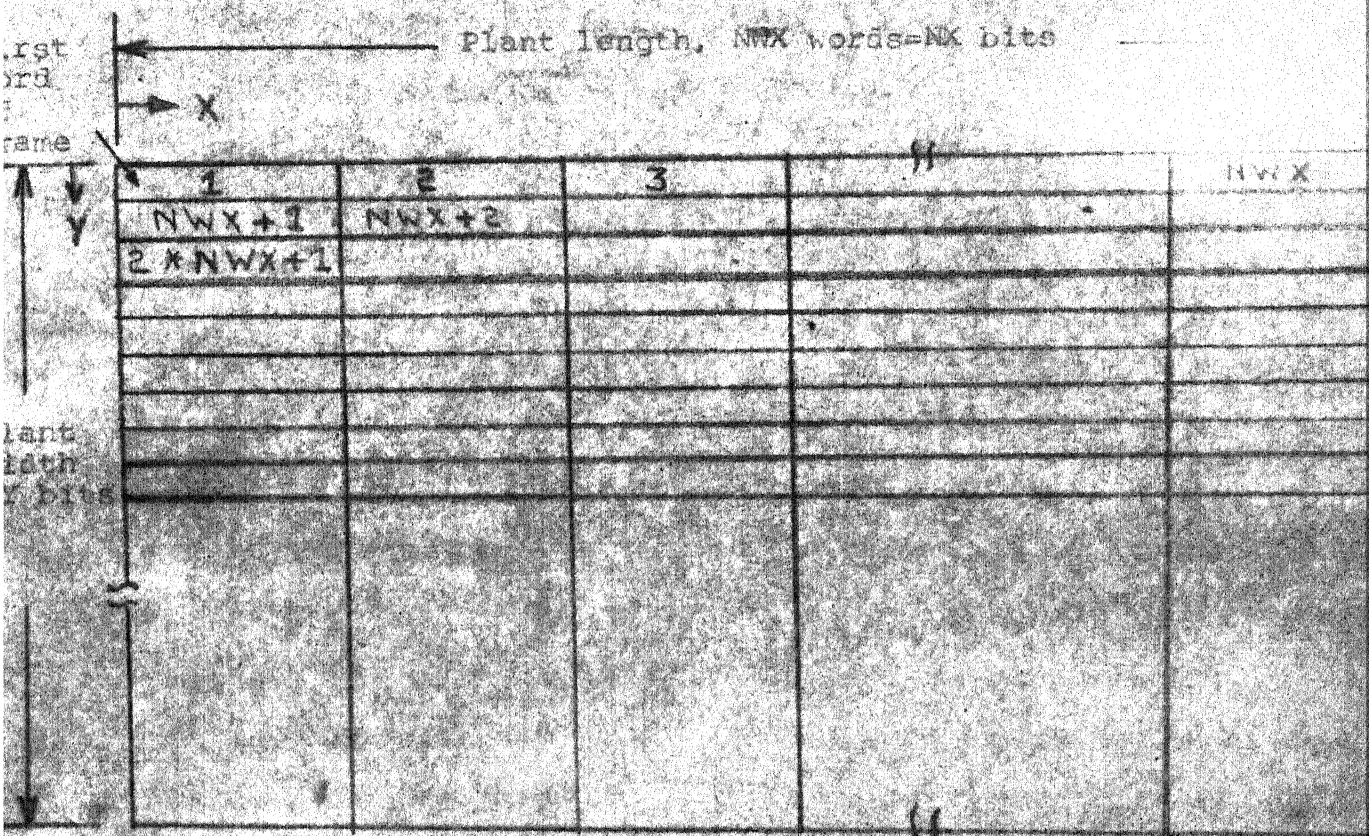
POINT (N) = (N-0.5) * PITCH

NBIT gives the bit number, in whose range a real value P falls. As an example, when PITCH = 0.1, both 4.5 and 4.59 fall in the range of the 46th bit. POINT is the return transformation, giving the point in real space occupied by a bit N. Since the bit space is discrete, the bits can occupy only a few discrete points in space.

The individual layers (bit planes) are all identical, and are of a standard format, for operational convenience. The length of the plant is aligned along the machine words (see Figure 1.1), and is composed of an integral number of machine words, NWX. The width of each layer, NY, is just enough to accomodate the plant width. It is initially verified, that for the specified pitch, the product of NY and NWX is less than 5000 (the number of words in each buffer). Else, the plant is too large, and the pitch may have to be increased.

While defining the objects, if a part (or whole) of an object lies outside the defined plant size, it is truncated at the plant limits. No message is printed out on these truncations. (A few other terms are explained in Section I of the Users' Manual).

The pattern matching method was decided upon over analytic methods for the following reasons: (i) Analytic methods tend to become complex with increasing number of objects and object shapes. A high price would have to be paid to retain the generality under all conditions. (ii) For checking each new object for conflicts, reconsideration of all the previously input objects would be necessary, and clearance calculations might become very complicated when dealing with obliquely oriented objects. (iii) The complexity further increases when some objects are defined in terms of



NWX = NINT(Plant length/36) word length = 36 bits
NX = NWX * 36
NY = NINT(Plant width)

The numbers in figure show the word arrangements.

F I G 1.1

other primitives. In the method adopted, bit patterns are generated, updated and stored for ever, still maintaining maximum flexibility in handling. The limited amount of hardware parallellism (in handling bits of a word simultaneously) is explited in speeding up operations. However, in the final stage of estimating the exact clearances, the analytical principles will have to be utilised to some extent.

1.5 References

1. Adolfo Guzman, "Decomposition of a visual scence into bodies", Project MAC, MIT, MAC-M-357, September 1967.
2. IIT-PAX Users Manual - K. Kannan and A.K. Shah, Computer Centre, IIT Kanpur.
3. R.G. Newell, "An interactive approach to pipe routing in process plants" IFIP Congress, 1971 (pp 46-50, booklet TA-6, August, 1971).
4. R. Galimberti and U. Mantanari, "An algorithm for hidden line elimination", Communications of ACM, Vol. 12, April 1969, p 206.
5. A.T. Aucott, "The use of computers in engineering services and construction industry", Heating and Vent. Engineer, Vol. 43, March 1970, p 437.
6. Robert J. Aubrey, "The use of computers by architects and engineers in the USA", Heating and Vent. Engineer, Vol. 43, April 1970, p 516.

7. Pipe Sizing by Computers - A report, Heating and Vent. Engineers, Vol. 42, September 1968, p 134.

8. Herbert Rosenthal, "Computer optimization of heat tracing design for process piping systems", Power, Vol. 115, August 1971, p 86.

9. "Piping Flexibility Analysis, Engineers' Check list", Building Systems Design, Vol. 67, July 1970, p. 61.

10. Howard M. Boyer, "Optimum pipe sizing by computer", Air Cond., Heating and Vent., Vol. 66, Jan 1969, p. 95.

11. John. E. Brock, "Automatic Digital computation of pipe flexibility", Piping Handbook, Fifth Edition, McGraw-Hill Book Co., 1967, p 4-79.

CHAPTER II

INPUT AND OUTPUT

2.1 Requirements

The input 'language' is to serve as an interface between the engineer and the program for conveying the descriptions of objects, the constraints, and other details and requirements. As such, it has to be unambiguous, yet generalized, logical, simple to learn and remember, and devoid of trivial restrictions. While being simple and fast for the more common shapes, it should be powerful enough for describing any not-too-common a shape with little extra effort. It is often worthwhile to improve upon these qualities of a language even at the instance of increased programming cost and complexity.

The output generated should serve as an effective feedback. It should have unique interpretations at any context, and should be complete and precise. Any particular detail that may be useful in verifying the input, understanding the execution and the results, or in planning the future runs, should be available to the engineer at his command. Clear indication of conflicts and their patterns is a part of the output.

A sophisticated input grammar with a tight syntax could only frighten the field engineer, who would be more at home with an input format that allows him to prepare data without being observed with too many syntax and construction rules. The needs and ease of the engineer should be the prime considerations in designing the input formats.

The performance of the keypunch operators is also an important consideration. The engineer seldom punches the cards himself in engineering organizations and the keypunch operator may not be reliable with an unconventional text. It is extremely important that mistakes in punching are easily detectable through a listing, and the input format structures could be of much assistance in this regard. Also punching reliability is higher with a simple and standardized format. Any punching or data error, over-looked by the engineer, should be evident in the output, and this requires that the output includes a complete reproduction of all the input data, with their interpretations.

Execution speed and ease of programming may also shape the input and output structures, till these do not adversely affect the more important considerations. All these considerations do not always act at cross purposes; an example is the simplification of input syntax and the consequent dispensing with an elaborate syntax analyser.

The input data structure should have enough interlocks and redundancy to help detect most of the possible errors in the data due to oversight or ignorance. Screening of all input data and verifying its construction is essential; the input structure should permit such a screening and the output must contain clear messages and instructions, whenever an error in data is detected.

2.2 Input Structure

Many versatile languages have been suggested in the literature. The one suggested by Guzman (1) is of particular interest. At the outset, a similar language appears to satisfy our requirements, with a LISP type structure in which an object may be defined using sub-objects, which in turn may itself be defined by other sub-objects, ultimately going down to a (sufficient) set of primitive shapes. Matching left and right paranthesis may be used to mark any object/sub-object, and a few composing operations, as OR, AND, etc may be allowed.

e.g.

```
OBJECT = (OR(AND(RECTANGULAR PRISM(x,y,z,a,b,c)),
                SPHERE (x,y,z,r)), CYLINDER (x,y,z,l,r))
```

The unlimited recursion that such a structure allows would make it extremely powerful and capable of defining unambiguously any shape. But such language has been decided against for the following reasons:

(1) Though the data structure is very logical, and resembles a natural language (making it easy to understand and use), the punching and checking efforts will be too much. Manual matching of paranthesis is a slow and patient affair.

(2) It would need an extensive lexical analysis by the program, which is bound to take up considerable memory area, plus execution time.

(3) All the versatility and power of such a language is not always necessary for defining the shapes normally encountered in a plant. It appears advisable to have built in definitions of the standard shapes as I sections, channels, beams, bracings, pipes, cylinders, etc., rather than expect the user to define them each time.

(4) Reading through a print out of a running format of the above kind, is quite difficult. It is bound to tax the user's patience and mistakes in the data may be overlooked.

It is felt that a language structure of this kind is more useful to a scientist rather than a field engineer. An engineer could feel more comfortable with a language structure that is faster and easier to read back, particularly since the volume of data is enormous and mistakes might cost dearly.

The above considerations have resulted in the selection of a standaridized fixed field input format. The definitions of all the common shapes encountered in a plant are built in and all standars are allowed to be defined before use,

extremely fast data preparation. And it still retains all the power of the above mentioned structure. The engineer is permitted to define new shapes (prevalent in the particular plant type being handled) by himself, in terms of primitives, and also use it repeatedly without defining it each time. A wide range of control cards enpowers the user to completely control the program and also get out of any tight spot.

The successive shaping of the language has taken place over repeated attempts at data preparation and it can be claimed with confidence that the maximum speed in preparing, punching and verification of input has been attained. The program logic itself has been designed with a view to simplify input, in all but a few instances, where the dependence has been reversed for pratical considerations. However, the effect of this has been in-significant.

2.3 Facilities

Besides the capacity to describe various objects, the input language allows certain other essential facilities and power to the engineer to control the execution and output and simplify his work. The following are a few facilities:

(1) Built in definitions of all standard shapes and objects, as columns, bracings, etc.

(2) Pre-definition of reference axes and standard I sections, so that these can be referenced in future symbolically.

(3) Pre-definition of non-standard shapes for repeated future reference - e.g. valves. There is also a facility to vary the scale of these shapes.

(4) Control over execution and output; the engineer is allowed to indicate the conditions under which execution is to proceed or stop, and the output required.

(5) Specification of minimum clearance required around objects and pipes.

(6) Besides adding objects, with or without check, there are facilities for removing objects also.

(7) A reasonable amount of retraction, to nullify possible mistakes.

(8) A complete trace of the execution, if required by the engineer.

(9) The horizontal section at any level of the plant is available on demand, to assist in planning the pipe paths.

(10) Conflict suppression at trivial points, as joints between pipes and equipment.

(11) Control over the program parameters, to allow changes in the configuration of the installation.

(12) Identification of the conflicting objects if required.

(13) Adjustment of the pipe path to avoid conflict, if required.

(14) Besides, there are provisions for recovering from any error condition, due to machine failure or accidents.

The actual data card formats and the input structure are specified in detail in Section 2 of the Users' Manual.

2.4 Guidelines

The development of the input data structure is a recursive trial and evaluation procedure, and it has to take place simultaneously with the program development. The design of input structure and the algorithms have to take place complimentary to each other, keeping in mind the comfort and requirements of the engineer. The following are a few guidelines adopted in the process:

(1) Minimum computation on the part of the user, all data being directly readable from the drawings available at the design stage.

(2) Data preparable in the logical order in which drawings are available; not more than one (or two) drawings to be referenced at a time.

(3) Data preparation and checking easy and fast; facility to pre-define all standards and frequent references, as I sections, channels, reference column lines, etc., standardized input formats, that are easy to grasp and remember. Maximum readability of input data.

(4) Establishing continuity between runs easy and simple; repetition of any detail to be completely avoided.

(5) Built in recovery procedures for minor machine errors, and simple restarting procedures for major ones, avoiding a re-computation.

(6) Efficient screening of input data for possible slips, without grave consequences. Demand a minimum redundancy in input for checking data that are too vital to allow errors.

(7) Maximum flexibility in the input sequence, and all details updatable or correctable anytime within reasonable limits.

(8) Effective data protection against user and machine errors. A minimum amount of redundancy in the data structure.

(9) Minimum use of machine dependent features to permit a change of system.

(10) Maximum assistance to the user and operator, in the form of unambiguous messages, outputs and instructions, to permit speedy understanding and further planning.

(11) Full control over execution of the program left to the engineer.

(12) Maximum computerisation of the steps involved in pipe layout design.

2.5 The Output

The output generated on the console and the printer serve the following purposes effectively:

(1) A convenient way of varifying the punching and data preparation, through a listing of the data deck.

(2) A complete indication of errors in the data deck that the program is able to detect, with a mention of the action taken by the computer and the actions to be taken by the engineer.

(3) A reproduction of all the input data, with their verbal interpretations, to assist in detecting mis-understandings or errors that have been over looked.

(4) Print outs of conflict patterns, horizontal sections and particular files in easily interpretable form.

(5) A minimum amount of status information to assist in restarting the program in case of machine failure or other accidents.

(6) Trace informations on the execution, if asked for, to give the engineer a complete idea of the process.

(7) Clear indications of error conditions, with the automatic recovery procedure adopted or the instructions to the engineer for restarting the program, as the case may be.

(8) Provide all details required to maintain the continuity between successive runs.

(9) Simple and complete instructions to the operator for all necessary actions, as tape mounting, removing, changing, etc.

-

CHAPTER III

DATA ORGANIZATION AND MANAGEMENT

3.1 In Core Data Organization

All the algorithms in the package are devised over the assumed availability of three buffers in core, each of 5000 words. These buffers form the reading in, processing and writing out areas for the tape operations. The provision of three such buffers, besides being ideally suited for most of the algorithms, also allows for extensive overlapping, through a modified cyclic buffering scheme (Chapter XII).

The buffers occupy together a major part of the memory - 15K out of the available 24K (after deducting for the system area). What remains is hardly sufficient for the program and other data both of which are considerably large. This calls for an overlaid operation (multi phase execution), so that much less space in memory would be enough to accomodate the program, in small enough segments. Since the operational efficiency goes down with increasing number of phases, the number of phases is to be minimised, the individual program segments ('core loads') being as large as possible. This requires an optimisation of the data area in core.

The normal optimisation procedures adopted in overlay systems are utilised here - (i) the amount of link information to be maintained between the phases, is kept at the minimum, and are placed in the common area (non-overlay area). When two sets of link information are operationally disjoint, they are equated in core by EQUIVALENCE statements, and this further reduces the size of link data area. Example:

Meaningful linkages	Core loads I and II	Core loads II & III	Core loads I & III	Core loads III & IV	Core loads IV & I
Link Data	A,B,C	A,B,D	A,B	A,B,E	A,B,E,F

Data C and D can be equated into the same locations in core. (ii) Within each core load, in each routine (or a closed set of routines), all such temporary variables that have no significance outside that routine (or set of routines) are listed together. The data items in each of these lists are completely disjoint, in that when one of the data lists is operative, the others are not being used. Hence these lists can be allotted a common space in the non-overlay area. Besides minimising the data locations, this also helps in completely compacting the individual core loads (as all the data area are extracted from these core loads). More routines can hence be packed into each core load. The idea can be extended using labelled commons, and table 3.1 shows a way of verifying the equivalences of data area.

Having allotted the major part of the memory to buffers, it is ensured that these buffers are completely utilised at any instant. The cyclic switching scheme maximises overlap during the execution of I/O bound algorithms. When there is not much I/O to warrant continuous buffer switching, the buffers are used for collecting information (as conflict patterns), before writing out - in effect, blocking the output records.

The length of each buffer has been fixed at 5000 words to provide the capacity to handle reasonably large plants (or parts of plants). With 180,000 bits in each buffer, a plant area of 1800 sq.m. can be handled with a reasonable pitch of 0.1M. The upper limit on the buffer size is fixed by the fact that three such buffers are to be accomodated in core.

3.2 Backup Storage-Problems

The least reliable components of any computer system are the Input-Output equipment. Failures of I/O equipment, tapes in particular, are more a matter of course than a rare accident. A package entirely based on input/output, as the one on hand, should have built in procedures to overcome any one of the so many possibilities of failure in I/O, with the assistance of the user or the operator if needed. This calls for a very sophisticated and reliable data base, with enough redundancy for error recovery.

Routines -	S1	S2	S3	S4	S5	S6
Labelled Commons -	A	A	B	F	A	A
	B	B	E		E	B
	C	D			F	E
	D					F

Prepare such a table, after the common allocations are over. The routines in all the core loads can be combined into a single table. A routine in the table may also represent a closed set of routines in a single core load.

Verification: Those routines, which share a common area, should

(i) be independent of each other (i.e. do not call each other)

OR

(ii) have the same interpretation for the arguments in the common area which they share.

e.g. S1 and S6 share area 'B'. They should have no linkages, or, they should have the same interpretations for variables in area B.

Table 3.1

Magnetic tapes pose a few characteristic problems which are absent in disk I/O. But a data base has been designed primarily for magnetic tapes, in view of their handling ease, normal availability and compatibility among various operating systems. A tape oriented data base is compatible for use with disk, but the reverse is not true. Further, the availability of a large portion of the disk at any instant cannot be assumed.

Magnetic tape devices of most of the systems share a few common problems. Foremost is their failure as a random access device in the 'read and write' mode of operation, i.e. it is not reliable to read a particular record from the tape, modify it and write it back exactly in the original space. This is due to the fact that the inter-record gap length, as well as the record length, are functions of the power voltage and frequency, and also vary from unit to unit. An attempt to update a record in this manner might result in nonstandard inter record gaps, or even elimination of these gaps, making the adjacent records in-accessible or erronous. The normally suggested remedy is to copy the whole tape length on a new tape, updating the appropriate records. This process might take upto 10 minutes for a 2400 feet tape, a big waste, and a crushing overhead, if a few hundred records are to be updated and re-updated in each run.

Bad physical handling of tapes and wrong mountings often spoil some records on tape, normally the first records. These records become irreteivable.

Also, there are many hardware limitations and restrictions. The IBM tapes (model 729), for instance, are not capable of detecting a file mark while backspacing. Further, a backspace command at load point does not produce a hardware error condition, and has to be detected by software. IBM manuals forbid a few sequences of operations with tapes. A direct change from a Read Command to a write command, or vice versa, without an intermediate non data transmitting operation, is said to produce stray signals in the reading station and hence erroneous transmission. There are a few other restrictions too (P.33, Form C28-6309-4, Input Output Control Systems, IBM 7040/7044 Operating System). All these effectively prevent the direct use of tapes in the random access mode.

Most of the error conditions in data transmission (detected by hardware) are due to malfunctioning tape units, dust and voltage fluctuations. Error conditions other encountered are redundancy in transmission and incomplete word transmission (transmission truncated in the middle of a word), and quite rarely, a transmission error (parity). Normally, a few repeated trials, and some tape cleaning (done by backspacing enough tape length, to bring current record under tape cleaner) result in a correct operation. In case of persistent

redundancy while writing, the IBM supplied package - Input Output Operations (IOOP) performs a recovery sequence, consisting of writing blank tape over current tape length and making more attempts to write. This again assumes that there could be no useful records in the tape beyond the current record, during a write operation. This is not compatible with the random access mode of operation.

There are besides, so many ways in which a tape device can fail mechanically, operator intervention becomes necessary and this often leads to a repositioning of the read/write head on the tape from where it was left by the program. It implies that before every Input/output operation, the current tape position cannot be assumed by the program, it has to be verified precisely, to prevent overwriting some other records. Data protection schemes are thus essential to make the data management reliable. The data organization should permit restarting the system after fatal misruns, power cuts etc. An optimum balance has to be struck between recovery speed and efficiency (which needs a good amount of redundancy) and operating speed (which is affected by the duplication of information transmission and storage).

All the aforesaid considerations have gone into the design of the data organization and management system. Inspite of the complicated requirements of such a system, a high degree of operating speed and efficiency has been achieved by cutting

down non transmission tape operations. This has been made possible by the fact that the algorithms and program on one side and the data handling system on the other were developed together, complementary to each other.

3.3 Towards Random Access

Ideally, the main data tape is to have a strictly 'random access' capability, in that any record can be modified and rewritten in its original place. As this is impossible, a slight variation of this has been arrived at. The new tape structure consists of regular file marks, throughout the necessary tape length. The distance between the file marks is enough to accomodate the record(s) to be read/written in that space, plus a slack space enough to take in the fluctuations in the record and inter record gap lengths. The files become the units in data handling. At any instance, the records within a file are either read in a sequence, or written in a sequence. The forbidden sequences of operations are carefully avoided. Updating is done file by file.

Since, unlike a disk, a search in a tape has to proceed in a strictly sequential mode, much time could be lost in searching and positioning of the tapes. With a view to minimise these, the current tape positions are always remembered and completely utilised by each routine, while hitting upon the ideal sequence of Input/Output operations.

The number of records per file is arbitrary. The algorithms used in the program require, for speed and economy of storage, an optimal clusturing of records. About five records per file has been found to be near optimum. This data structure, obviously, needs an initialisation of the tapes with files, each file containing the required number of records, the record gaps and the slack space. The slack tape length in each file should be enough to accomodate all fluctuations. Each file contains the patterns corresponding to a 'group' of layers of the plant.

3.4 Tape Functions

The program speed is entirely dependent on access speed, and the assigning of functions to various tapes is done with the sole aim of minimising the access delays, besides simplifying the updating algorithms. A bad tape assignment might lead to a condition when the majority of the execution time is wasted in searching and rewinding tapes. Maximising the scope for overlapping is another useful consideration.

These necessiate that the tape assignments and the program I/O strategies and sequences should be designed to help each other. For purposes of speed, ease of overlay and easy error recovery the program execution has been identified into a few disjoint phases, for each 'data section'. The execution proceeds section by section and within a section-phase by phase, sequentially.

A minimum of five tapes is necessary for the operation of the program. The contents and function of the various tapes are tabulated in Section U-V of the Users' Manual and are further described in Sections 3.6 to 3.8.

3.5 The File Table

The 'File Table' is a single dimensioned array, containing the file number corresponding to each set of N contiguous layers (a group) of the plant, starting from the base of the plant. N is the number of layers per group, = number of effective records in each file (ignoring the header and the trailer records). The file number itself contains the tape number also, and is computed as

File Table Entry = tape number×1000 + file number

The file table in core gives file numbers of the most recent version of the groups of layers. For instance, the L-th entry in the file table gives the file number at which the patterns at layers [(L-1) * N + 1] to [L*N] exist in the most recently updated form. Example: if N=5, and the fourth entry in the table is 1007, it indicates that layers 16 to 20 (group 4) of the plant are in tape 1, file 7. The base plane has layer number 1.

The size of the file table fixes the maximum number of groups possible, and hence the plant height. It can be varied at system assembly. For a given table size, N is

fixed automatically, such that

(i) N * table length $\geq$ total number of layers in the plant [= NBIT (Height/Pitch)]

and (ii) N $\geq$ 5

The file table forms part of the 'snapshot', 'store' and 'restore' areas. It is storedduring a normal termination, and restored at the start of the next run.

3.6 Main Tapes Structure

As mentioned earlier, the main tapes have a filed structure. N is the number of effective records per file, normally fixed at 5. It may be varied during system generation, if necessary, and later by the program itself, in case the file tabl is too small to accomodate the plant height.

Two extra records exist in each main data file. The one at the start, numbered record 0, is a 3 word header record (or status record), and contains the following information:

(a) A number NLEVEL indicating the group number residing in this file. A value L indicates that layers [(L-1) * N+1] to [L*N] are in this file.

(b) The previous file number, from which this file was obtained by updating (=NPRVS).

(c) The stage number, when this file was written (=NSTG)

Any particular file is not rigidly assigned to a particular group. The assignments vary each instant, and are available in the file table.

Overwriting of files is delayed to the maximum, and the most outdated file is chosen for overwriting, from the list of outdated files. (Refer updating strategies - Chapter VII). These and various other steps ensure positive recovery or reconstruction after any type of error condition.

The other extra record in each of the main data files (the trailer) consists of information about the objects present in that set of layers, and these informations are utilised later in identifying the conflicting objects (Phase V). There is also a parameter NNEXT, which if non-zero, gives the next updated version of this file. If it is zero, it indicates that this file contains the latest version of the group.

It may be mentioned here that not all files are updated in Phase III, for each 'section'. Only the files that are affected by the particular section are updated.

3.7 Auxilliary Tapes

The auxilliary tapes mainly serve as scratch tapes for the intermediate outputs between the various phases and for taking 'snapshot' of the program status at particular instants for error recovery. Snapshot is essentially a store operation, performed more frequently. Besides, the tape AUX1 contains the overlay core images and the non standard object descriptions. These are assigned to the Auxilliary tape mainly as a time saving measure, in view of the high frequency of usage of the overlay core images. If they were placed on the main

tapes, a long rewind will have to be performed each time a core load is to be swapped in.

The start of the third file on the tape AUX1 is variable. Each time a new object is defined, the current file mark is overwritten and a new file mark is written at the end. As these object definition records are not updated, no slack space is necessary at the end of the file.

The main and overlay core images are stored in duplicate to prevent a breakdown if some of these records get damaged. Creation of this tape is described under 'System Generation', in the supplement.

3.8 Program Tape

The 'program tape' contains the up-to-date listing of all the input cards fed to the program. Alongwith are stored the run numbers, section number and card numbers of the card images and sections and the results of the executions. A listing of this tape can be obtained on demand, and serves as a record of all the activities to date. The information on this tape are also used in the identification of the conflicting object.

During any particular run, the data deck is first read in till the first JOBEND card and written on this tape commencing at the suitable point, concurrently producing a listing on the printer. The tape is then backspaced to the current data deck start and exeucition starts. Under certain error conditions (due to machine failures), a repeat execution

of the current section is performed by backspacing this tape to the start of the current data section and restarting. The tape is to be saved between runs. Most of the automatic recovery procedures utilise the contents of this tape in reconstructing the lost links and patterns.

3.9 Labelling Scheme

A compact labelling scheme forms the back bone of all the data protection and error recovery procedures of the program. It helps in judging the current position of the tape, respositioning of the tape, and in verifying each I/O operation after its completion, to be sure that it has been done at the right spot of the tape. The labelling scheme also helps detect certain conditions as wrong mounting of tapes, request to print out non existent records, etc.

The label consists of a single word, that contains the tape number, file number and record number of the particular record. All the records in tapes MAIN1, MAIN2, AUX1 and AUX2 are labelled. The format of the 36 bit label word is

Bits 0 to 5 - Tape number
Bits 6 to 20 - File number
Bits 21 to 36 - Record number

The tape number is the FORTRAN logical unit - normally, the single digit number to which the unit is dialled (unless some switching has been done).

The first word of each labelled record forms the label word. To make the labelling meaningful, the numbering of the files and records on the tapes are done strictly in the sequential order.

3.10 Data Protection

Ignoring the occassional errors that result in destruction of part of the vital data may prove to be too costly as it might involve a complete recomputation running to a few hours. Embedding all input/outputoperations with a few checks and locking mechanisms would help eliminate harassing conditions, as destroying the results of a few hours' computation by a single misplaced write operation. The data protection mechanism essentially consists of verifying the legitimacy and correctness of each I/O operation, and its position on the tape. Special care is given to write operations, to check that vital data are not overwritten. The facilities gained by the labelling scheme are exploited.

Too tight a protection mechanism might very much increase the execution time, due to redundant operations. Hence the barest minimum protection is provided. However, even in the very remote cases when the protection might fail, simple error recovery procedures (automatic or manual) are available whether such occurances are detected immediately or at a later stage.

Reading of particular records do not pose a protection problem, as the tape records are not altered. However, all read operations are thoroughly checked - the label of the record is matched with the label computed, and the record length (number of words transmitted) is also checked. The IBM supplied error recovery procedures are used for error conditions,

consisting of upto about a hundred trials interspaced with tape cleaning operations till an error free reading has been achieved. In case it fails, a standard error recovery procedure is taken (and printed out), depending on the context and seriousness of the condition. If the labels do not match, the program repositions the tape to the required file and record, as it knows the data structure. However, if even the tape numbers do not match, suitable messages and instructions are printed on console.

Checking the write operations consists of positioning the write head at the exact point, and performing an error free write operation. Positioning is done by reading the current record with its label, and moving the tape in the right direction. It is then checked that the right record has been reached by reading the label. A backspace operation and a write operation follow. The new record overwrites the old one, with the label part unchanged. If an error condition results, a backspacing of four records is made and the process repeated. After five trials this way, five more trials follow with a forward spacing of four records, and a fresh attempt to write. If the error persists, the current file is written off and a new available file is written in.

The standard IBM recovery procedure consists of two trials at writing, after which a blank tape is written over a length of 3.5" and attempt to write is made afresh. This procedure is suppressed, as it might result in overwriting of adjacent file marks.

When a few sequential write operations follow in quick succession, the first writing is checked out completely, and the second and further write operations are not checked for the tape position. However, if an attempt is unsuccessful, all further attempts are thoroughly verified. At times, the writing of a record may make the next record (and its label) irretreivable. Care is taken to make sure that this does not create problems (i.e. reading the next record is avoided).

Extensive trials to date have consistently shown a positive recovery in the second trial of I/O, if the first trial fails. No serious problem has been encountered.

3.11 Redundancy and Retractability

Inspite of the data protection mechanisms, there may be instances when some of the records get spoilt, due to physical mishandling of tapes or otherwise. In such cases, recovery procedures have been devised, involving the retraction of a few steps and building up again. This is made possible by the fact that atleast one set of previous files is always available at any instant. Also, as mentioned earlier, the overwriting of files is delayed to the maximum, and the most outdated file is always chosen for overwriting. For error recovery, unless the damage is enormous, a retraction by one 'stage' would be enough. More so because the files of adjacent stages reside always on different tapes, without any possibility of their both getting damaged (Chapter VII).

This facility to retract besides being useful in error recovery, could be useful to the user also. It could be a very fast way of undoing the last few sections, if extensive errors have been committed in them. (The GOBACK card does this operation). No new computation is involved and only the 'Files Table' is to be reconstructed, using the chaining information available in the current files. If the current file has not been changed since the stage to which GOBACK is requested, no alteration in the table is necessary for that file.

The degree of going back possible essentially depends on the amount of redundancy. With the minimum permitted value of 50% redundancy (i.e., twice the minimum number of files) a degree of GO BACK of not less that one stage is possible. This will be actually much higher, since not all the files are updated at each stage.

3.12 Level of Input/Output Routines

The input output routines are written at the IOOP level (Input Output operations of IBM 7044). This has been selected in preference to the more sophisticated IOBS (Input Output Buffering System) for the following reasons:

(1) The data area for I/O operations is very large, of the order of 5000 words and IOBS would require extra buffers at least as big. IOOP permits the use of data area itself as the I/O buffer. Further, in IOBS, there is an overhead in

the transfer of data in core to the standard buffer area, before actual transmission.

(2) Non-standard error recovery procedures are possible only with IOOP.

(3) Complete control is available over the over lapping of operations in IOOP.

CHAPTER IV

THE PROGRAM LOGIC

This chapter provides a broad outline of the general strategy adopted and forms an introduction to the more detailed discussions in the chapters to follow. As such, the details here are rather disjoint, but an understanding of them would be essential for appreciating the future discussions on program strategy.

4.1 The Basic Routines

The whole program has been knit over a few routines, part of them in assembly language that perform the basic functions. As these routines are extensively used by all phases of the program, their reliability, power and speed are critical considerations. A complete discussion is available on a later chapter on the 'Algorithms' (Chapter IX). A brief mention on their functions is provided here.

(i) Insertion of Rectangles - given the frame, and its dimensions and the rectangle parameters in plane coordinates, bit pattern of the rectangle is created in the frame. The variations possible are: (a) adding the rectangle, (b) adding the rectangle, checking for conflicts, (c) removing the rectangle, (d) checking while removing, that bits to be removed are all on before removing.

(ii) Superimposition of frames - and the extent of superimposition required, the two frames are superimposed and result placed on the specified frame. The four variations mentioned for rectangles is available here too.

(iii) Tape operations - A routine takes care of all the tape operations, including the checking and error recover procedures. Parameters to be specified are the tape number, file number, record number, starting address and word count for data transmission, whether pre-positioning is necessary, and the degree of checking required.

(iv) Position transformations - Conversion of symbolic position references (with respect to defined axes) to absolute values is done by a routine by searching the axes table for a match. Search can be ordered along only X or Y axes or both. Given the limits of the extent of the object along a direction, and the physical limits of the plant being considered, another routine determines the bit limits of the object using the function NBIT. The return transformation from bit space to real space is done by the function POINT.

(v) Print out of a record - Given a frame of pattern, the routine prints it out in a readable format, within the limits specified. All 'on' bits are printed as 'X', and off bits are left blank. The reference axes are also printed along the borders. A variation of this routine is used in printing out conflict patterns, which, given two frames, the common patterns and the individual patterns are printed out in different shades (letters).

4.2 Control Cards

The actions taken when the normal control cards are encountered are discussed here. The actions for error recovery control cards are discussed alongwith the error recovery procedures at a later stage. So also, the functions of the SYSGEN cards are discussed under the overlay monitor.

(i) First Card - The first card is read in and immediately checked for validity. If it is an error recovery control card (REBUILD, MAKEPRO, etc), the suitable actions are taken (as discussed in Chapter XI), and after the standard recovery procedures, a RESTORE card is assumed in its place. The first control card possible under normal conditions are START, RESET, RESTORE or a LISTPRO. A SUMMARY card, if present, specifies that a summary execution is to be performed, and it should be followed by one of the above cards essentially.

Whenever a tape has to be restored from the previous run to the present run, the first operation performed on that tape is, by principle, a read operation. This allows a checking of the correctness of the tape mounting. The tape AUX1 has the program coreloads, and is hence checked in all the cases. For START, the Main and NPRO tapes are to be created afresh and hence are not checked. For RESET, the NPRO tape is to be created afresh, but the existing field structure in Main tapes is to be utilised for the current problem, and

these two tapes are checked. LISTPRO is similar to RESTORE, in which all the tapes have to be restored from the previous run.

Following this, the current data deck is listed on the printer and NPRO, simultaneously, till the first JOBEND card is encountered. The tape NPRO is initially positioned at the end of the last data deck (in the case of RESTORE, LISTPRO). After the listing, the tape NPRO is respositioned to the first card of the current data deck, and all future data are obtained from this tape.

For a RESTORE,LISTPRO, the basic data, as the file table, definitions, etc. are restored from the store/restore area of Main tapes and normal execution commences. For START/RESET, the basic data, as the plant dimensions, pitch etc., are read in from the basic parameters card (that follows, essentially), and the size of each frame is computed. It is verified that, for the given pitch, the plant layers can be accomodated within 5000 words. Next, it is verified if the length of the file table is enough to accomodate the plant height, for the present value of N-the number of layers per group/file. The value of N is obtained from the Restore area, in case of RESET, and is assumed (as defined during system generation), in case of START. In case of RESET, if this value of N is workable (for the given file table size), no initialization of the Main tapes is necessary. Else, as in the case of START, the main tapes are initialised, after computing

the new value of N that satisfies the test (3.5).

If the tapes are initialized, the total number of files available on each of the main tapes is determined. In case of RESET, when the tapes are not be initialized, these values are available in the Restore area. It is verified that each tape by itself can accomodate at least one copy of each layer of the plant - this would ensure that there are atleast twice as many files, in the two tapes combined, as the minimum required for storing one copy each of the layers of the plant. If any of the three tests above fails [namely, the frame size, the file table size, or the tapes size], the execution is terminated with suitable messages.

Next, a STORE operation is performed, that overwrites the previous data (if any), in the store/restore area. Also, the through columns area (record 0, file 0 of MAIN2) is initialized to all zeroes, to indicate void. The file table in core is initialized, such that all entries point to this record of through columns. Thus, the work is simplified in case of a RESET, in that even though the individual files throughout the tape may contain residual patterns, they need not be wiped out to zeroes.

Before the actual execution starts, all these basic data area printed out, for the users' information.

(ii) GO BACK - The file table gives the present file position for each layer. The stage number upto which records have been erased is maintained by the program, and if a GO BACK to an earlier stage (for which records are not available on tape) is requested, execution stops with a message. If the GOBACK is possible, it is done level by level from the base. For each group of layers, the tape is positioned to the header record of the file indicated in the file table. The stage number when these layers were updated, and the file where their previous version could be found are available in this header. If the stage number is less than or equal to the GO BACK stage requested, this file number is entered in the file table. Else the header of the previous file is accessed and the procedure repeated.

During a GOBACK, the shapes and standards defined are not nullified. This facility is not realy necessary.

(iii) SUMMARY - When this card is used before all the cards, in the current deck, it is ensured that none of the tapes are written upon. The execution is only a summary one, to point out errors and inconsistencies, and the user is frequently reminded of this fact in the print out. In the end, the tapes would be exactly in the same status were left in the previous run.

4.3 Sections and Subsections

A 'section' is the data segment between a complement pair of section delimiter cards (start and end). A section is the unit of execution, in the sense, execution proceeds section by section. For convenience some types of sections are composed of subsections.

Sections that actually place an object (objective sections) are assigned a 'stage number', in the sequential order of occurrence. This stage number plays a vital role in the data maintenance, error recovery, object identification, etc. The stage number assigned to each objective section is printed out at the end of the section for the information of the user. The user has to quote this number when using the REBUILD or RESTART cards. Section that predefine standards or object shapes are not however assigned any stage number.

The user has a further significance in the section-subsection structure, in that objects falling within a section are not checked for conflicts against each other. This facility is useful in avoiding conflict indications at pipe joints, for example. Also, minimising the operations on main tapes ensures a greater flexibility of operations, in that more of the previous versions of the various levels are available any time. Hence it is advisable to make the individual sections as large as possible.

4.4 General Execution Strategy

The execution of each section, in turn, proceeds as a multiphase operation. The splitting up of the execution has been done with the following aims -

(a) to standardise operations with various types of objects and thus minimise programming effort and complications. without compromising speed to any appreciate extent.

(b) to permit execution in a strict multiphase mode, with the individual phases overlaying the same programs area (Chapter X). Judicious identification of phases greatly minimises number of overlayers.

(c) to permit, in turn, over lapped [illegible]. Since multiphase execution demands much less program space, more buffer areas are available in core, permitting overlapped input/output/computation.

(d) Separating the execution of each section into distinct phases permits much faster and reliable error recovery, since it is enough that execution is commenced again from the current phase, rather than repeating the whole section. This very much improves the program performance and reliability.

(e) To induce a better understanding of the execution in the engineer through the standardization of the procedures.

Minimising tape operations has been a major consideration in identifying the phases, and the procedures adopted within the phases. It has been ensured that (1) most of the time, the tapes are ready at the right position, (2) all tapes are rewound simultaneously, or during other computations, minimising the waiting time for the processor, (3) majority of the I/O are done in parallel with computation, (4) in case of error/failure of any magnitude, it is always possible to recommence execution from the current phase; i.e. the data needed for the current phase are available till that phase execution is complete. To achieve this, a snapshot is taken at the end of each phase, and the user intimated.

The various phases of operation are as follows:

Phase 1 - Creation of bit patterns for the object, along the plane of the object and storing them in AUX2 alongwith other necessary details for phase 2. If necessary, the section is split into many segments, each segment having an unique plane. There will be one record per each such segment on AUX2.

Phase 2 - Extraction of horizontal sections along standard planes horizontal bit/from the phase 1 output. These horizontal sections are stored sequentially on AUX1.

Phase 3 - Updating the concerned records in Main tapes (affected layers), with the phase 2 output. It is essentially superimposition, checking for conflicts if asked for. The conflict patterns are sequentially stored on AUX2.

Phase 4 - Generating conflict pattern printouts, depicting the original object, the new object, and their overlap in different shades. These are generated from Phase 3 output plus the Main records and AUX1 is used as a scratch tape.

Phase 5 - Identification of the conflicting object - its stage number, card number and identification code; computation of exact clearance in critical cases.

Phase 6 - Adjustment of pipe path, if asked for.

4.5 Pre-definition Sections

As mentioned earlier, the sections predefining standards and object shapes are treated with a difference; they do not undergo the multiphase execution, being of relatively less complexity. The programs dealing with them all reside in one or the other of the core loads, depending on availability of space.

(i) Definition of Axes: The table of defined axes ('column lines') are updated. If some of the axes have been defined already, the new definition is taken, with a message. The table sizes fix a ceiling on the number of axes, and it can be varied during system assembly if required.

(ii) Definition of 'I' Sections: The table of defined 'I' sections - their name, web and flange - are updated. In case of multiple definitions, the last definition holds. Again the table size fixes the maximum number of sections that can be defined, and can be varied at assembly time.

(iii) Definition of Object Shapes: The multiphase operation is utilised to advantage in storing bit pattern for non-standard shapes, in a form suitable for fast handling. The phase I operation is performed for these shapes, i.e. bit patterns, are generated along standard planes for the object, compacted, and stored in file 2 of AUX1, alongwith other data for Phase 2. When using these shapes, the new orientation and scale are to be provided. These informations are utilised to modify the relevant information and these modified records are copied on to AUX2, as the Phase 1 output for the object. Rest of the procedure is similar to that for normal objective sections.

-

CHAPTER V

PHASE I EXECUTION

5.1 Outline

Phase I consists of (i) reading in the data cards from the data tape, checking the format and screening the data alongwith, till the end of the section is reached, and (ii) simultaneously generating the corresponding bit patterns of the objects along planes that are significantly related to the object orientation.

Each type of format has typical characteristics, which are tested for when the cards are read. If these are absent, the card is analysed to sort out the possible error. A message is printed out alongwith the action taken. In the next phase of screening, further checks are made on the validity of each data in the card - e.g., a wrong orientation (other than X or Y), use of an undefined axis, etc. If any discrepancy is found, suitable actions are taken and printed out. All the data are reproduced in the printout, with their interpretations.

In case of summary execution, or error conditions which result in summary execution of sections following the erroneous section, the execution is stopped for this section with the screening phase. For normal execution, the execution proceeds further. Unless the error is fatal, attempt is made, in the

given order, to (i) give a warning message, ignore/assume suitable values and continue execution, (ii) perform a summary execution of the current section and proceed normally with the future sections. (iii) perform a summary execution of the current and all future sections - this mode is chosen, if the previous mode might increase execution time too much, e.g. insertion of columns, which affect all the levels. If the error is fatal, the execution is stopped immediately with a normal termination procedure. An example - when the section has not been properly ended, and it becomes impossible to separate the two adjacent sections in a positive manner.

With this screening of input data, the bit patterns are simultaneously generated along parallel planes, parallel to the plane of the object - these planes are chosen, internally, to minimise computation, as well as to generalize the procedures for various types of objects. The object(s) within a section are split into groups/segments if necessary, such that each such group has a distinct object plane.

These bit patterns are sequentially stored on tape AUX2 starting from the load point. Each record consists of 5050 words; the first 5000 words contain the bit patterns, and the next 50 words the information that would help in the interpretation of the patterns (in Phase II). Each record (the first 5000 words) in turn consists of one or more equal partitions, also called frames. Each frame consists

of the bit pattern of the object along one of a few parallel planes (parallel to the object plane). To minimize the number of tape operations,

(1) As many frames as possible are packed into one tape record. For this, the frame size is chosen as the minimum required to just enclose the object.

(2) Redundant frames and repetitive frams (i.e., frames with same pattern) are avoided, and are suitably indicated.

(3) The object(s) within a section are grouped into as large individual groups as possible, such that the object(s) (or part of the object) within a group share the same object plane. Each such group produces one (or more, if necessary) record on AUX2. Minimising the number of groups minimises the tape operations. (These are soon explained in greater detail for the individual cases).

Depending upon the orientation of the object plane, the records are classified into four types:

Type 1: Object plane is horizontal

Type 2: Object plane is vertical, parallel to YZ plane

Type 3: Object plane is vertical, parallel to XZ plane

Type 4: Object plane is oblique (other than types 1,2,3)

The complexity, and hence the computation time increase down the list. An object is often split into pieces, such that minimum of the object has to be classified into the higher order types, if at all necessary. The gain in speed for lower order types is achieved in Phase II, using the parallelism in bit handling operations.

The extra 50 words, that form the key for understanding the patterns in the record are called the Link data, and consist of the following information:

1. NTYPE = Type number classification of the record, 1,2,3 or 4.

2. NZLOW, NZHIGH = Lower and upper limits of standard horizontal bit planes affected by this record.

3. NYLOW, NYHIGH for type 2 = The lower and upper Y bits, to which the 'frames' in the record (all parallel to X2 plane) are synchronized.

   NYLOW, NYHIGH for type 3 = Same as above, but the X bit limits, as the 'frames' are parallel to the YZ plane.

   NLOW, NHIGH for Type 4 = The limits of the frame numbers.

4. NPART = Number of frames in the record (partitions)

5. NWORDS = Number of words occupied by each frame (all frames of same size).

6. NPX, NPY = The dimension of each frame, the No. of bits along each side. NPX is always a multiple of the machine word length (=36 bits), and NWPX = NPX/36 = the frame width in number of words.

7. NWXO = The X coordinate of the frame origin, expressed in word number (significant only for types 1 and 2). The X bit of the frame origin is always synchronized with the first bit of a word.

8. NYO = The Y coordinate of the frame origin, expressed in bit number. Significant only for types 2 and 3.

9. N = Total number of frames (in type 4). This gives an indication if next record on tape is a continuation of the present one.

10. Four words, that identify the object to which the patterns [illegible].

11. A set of other parameters, for type 4.

12. Composing operation for this record (OR, AND etc), only for non-standard shapes.

The significance of most of the parameters above will be clear in Chapter VI. This chapter will deal with the computation of these parameters and the bit pattern construction for various standard shapes.

In type 1, if NPART = 1, but NZLOW ≠ NZHIGH, then the only frame in the record is to be inserted into each of the standard horizontal planes between NZLOW and NZHIGH. Else, NPART = NZHIGH - NZLOW + 1, indicating that frame 1 goes into standard bit plane NZLOW, frame 2 into standard bit plane NZLOW + 1, and so on.

In type 2, if NPART = 1, and NYLOW < NYHIGH, the only frame in the record is to be inserted repeatedly into all the vertical bit planes between NYLOW and NYHIGH. Else, there is one frame per vertical bit plane, in the ascending order. The same holds good for type 3 records; only the orientation of these vertical bit planes differs.

In types 1, 2 and 3, the plane of the frames is synchronized with the natural bit planes of the standard bit space. This ensures that each frame affects only one bit plane. All the frames in a record share a common origin, i.e., for a type 1 record, the frame origins coincide in plan;

for a type 2 record, they coincide in front elevation- and for type 3 in side elevation. This origin is also synchronized with a bit in the standard bit space. All these ensure that any particular bit in any frame (for types 1,2,3) coinides exactly with one bit in standard bit space; and hence affects only that bit. If these precautions are not taken, the object dimensions would tend to increase in phase 2, when these frame patterns are transferred to standard bit space, since in view of our safe sided approach, each bit in frames would turn on upto eight bits in standard bit space. [Because of the complete lack of synchronization of the cubes, each cube in the frames would fall into the range of eight adjacent cubes in standard bit space.]

Further, by fixing the X coordinate bit number of the origin for types 1 and 2, to the first bit of a machine word Phase II can be made extremely fast.

To summarise, the various steps taken ensure

(a) Maximum core utilisation

(b) Minimum number of records and hence tape operations

(c) Minimum 'Noise' generation (in Phase II)

(d) Maximum speed, exploiting all the available parallellism in bit handling.

However in type 4, due to the complete obliqueness, not much can be gained in speed or noise removal by any kind of synchronization of origin. A fair amount of speeding up is achieved by fixing one edge of the frames horizontal, and synchronized with a standard horizontal bit plane. (This is

always possible, though the resulting frame size may not be the minimal one.) Further, this horizontal edge of the frame has its length a multiple of the word length. These two steps reduce noise and improve the speed of Phase II to some extent.

To generalize the details given so far, in all the four types of frames,

(i) The frames are all rectangular

(ii) One edge of the frames is always horizontal and synchronized with a standard horizontal bit plane

(iii) This horizontal edge has its length a multiple of the machine word length (36 bits)

(iv) Both edges of the frame have just enough bits to cover the object.

When a single record is not enough to contain all the frames, extra records are created, with suitable modifications in the link data.

Also, the individual objects (or their parts) loose their identity after phase 1 is completed. In further phases, no discrimination is made on the type or identication of the objects. However, phase I procedures for the various object types vary widely and are discussed individually in the sections to follow.

In case of predefinition of object shapes, the Phase I output is written on AUX1, file 2, for future references. Execution stops with Phase I. When these predefined shapes are called, the corresponding records on AUX1 are copied over to AUX2, with slight modifications in link data and execution commences from Phase II.

5.2 Columns, Beams and Floors

It may be observed, for these three types of objects, that

1. The most convenient object planes are the horizontal planes.

2. They extend over the complete plant, and hence the individual frames are almost the same in dimension as the standard horizontal bit planes.

Incidentally, they conform truthfully to the normal Phase II output. Hence it is not necessary to have two distinct phases for these objects. The data screening and the bit pattern construction of Phase I are merged with the writing out portion of Phase II. The algorithms are discussed in Chapter VI.

5.3 Bracings

The most convenient planes for handling bracings are the vertical ones, oriented parallel to X or Y axis (i.e. type 2 or type 3). The size of the frames are fixed to the plant limits, and are not optimised. The vertical edge is fixed at the plant height, whereas the horizontal edge is fixed at the plant length (for type 1) or plant width (for type 2, rounded to complete machine words, as the plant width need not be a multiple of word length).

The gudget plates used for connecting the bracing arms at the centre and corners are inserted automatically at appropriate points. These plates are assumed square, and their size can be specified by the engineer. The gudget plate at the centre is always inserted, whereas the plates are inserted only at those corners where an arm exists.

Each data card defines the span, the plane and the top and bottom limits of a bracings 'frame' (the rectangle enclosing the bracings), the width and lateral thickness of the bracing arms and a mention of which of the arms are present. More than one card may define a single bracings frame when three arms are present, or the dimentions of the arms vary.

From these details, the type of the bracings plane (=2 or 3), and the limits of standard vertical bit planes that are affected, are computed for each card. For the first card, a frame of the required size is prepared, and is initialized to all zeroes (void). The patterns of the arms and gudget plates, defined by the first card, are inserted into this frame at the exact points. The bracing arms are essentially treated as rectangular prisms, and the patterns inserted in the said frame are their projections on a vertical plane - an inclined rectangle of width equal to that of the arms, for each arm. The plates, as mentioned are squares of the given size.

The second card is read next and the type, and limits of vertical bit planes affected are computed for this card. If the type and the limits of planes affected for this card are exactly the same as those for the previous card, the new bracings pattern is inserted in the same frame. If not, the old frame is written out on AUX 2 alongwith the link

informations, . and a new frame of size corresponding to the new type number is prepared and initialized to zeroes. The pattern for the current card is then inserted in this new frame. This procedure continues till the section ends. When the end of section is reached, the last frame is written out.

In view of the above algorithm, the following steps ensure minimum execution time, by minimising the number of frames written out. All the bracings in the same vertical plane should not be separated and within a group of bracings occupying the same vertical plane, those with the same lateral thickness should be clustured together. This grouping can be made automatic, with the program selecting all such cards from the section by itself. But this will necessiate too many passes over the same section; the data cards themselves are on tape, and the number of various planes which the bracings occupy is so enormous, that such a multipass operation may prove to be time consuming.

To achieve overlapped operations, whenever a frame is to be written out, the output command is issued and the new frame is initialized in a different core area ('Buffer'). The frame initialization and insertion of patterns are done even while the frame is being written out.

It should be noted in the case of bracings, that the same pattern goes into each of the standard vertical bit planes affected. No discrimination is made between the plates and the arms, and even though they may have different lateral thicknesses, they are treated as equally thick. Hence in the input data, the lateral thickness should be the overall effective lateral thickness, to be on the safer side. Further, the given lateral thickness is assumed to be equally distributed on either side of the plane of the bracings. If this is not so, suitable corrections should be made in the overall thickness.

5.4 Pipes

A data section defining a pipe network is composed of a few subsections. Each subsection defines a contiguous piece of pipe through all its bends, gradients and diameters at various points. This piece of pipe is completely defined by its diameter and its radius of curvature at each significant point in the pipe path, travelling along the pipe. Such a significant point could be (1) the starting point, (2) a point where the pipe bends, called an intermediate point and defined as the point of inter-section of the pipe centre lines before and after the bend, (3) a point where the diameter of pipe changes or (4) the ending point; changes in pipe diameter are considered to be abrupt.

It is checked by the program that the distance between two points in the pipe path is enough to accomodate the bends of the pipe at the two points. If not, execution is stopped with a message (Figure 5.2).

Each card in the subsection defines one significant point, Xi, Yi, Zi, Ri, Di (Figure 5.1). The start and end cards give the terminal points and the intermediate cards the intermediate points.

It may be obserted in any plant that the majority of the pipes are run parallel to the axes, on grounds of simplicity of layout and concerned calculations, minimum 'apparent' occupation of space by the pipes, and also on aesthetic grounds. However at tight points, a completely oblique pipe may be unavoidable. Hence the most general treatment is required for pipes. To combine speed with generality, an internal 'segmentation' procedure is adopted, to divide the given pipe into various segments in such a way that overall execution time is minimised. The pipe is segmented such that each individual segment can be accomodated in a plane. Since three points uniquely define a plane, a segmentation as in Figure 5.3 ensures that the individual segments (Case A in Figure 5.3) always lie in a plane. It should be noted that only the centre line of the pipe is considered for coplanarity. Once the planes of these segments are determined, the phase I execution consists of constructing the bit pattern of the pipe along a few parallel planes, placed uniformly apart with

an interval = 'pitch', each plane parallel to the centre line plane of the pipe segment. Each of these pipe segements will fall under one or the other of the four types of planes mentioned earlier.

A fair improvement in performance is achieved by adding another criterion for segmentation. It is, that, the segmentation is to be performed in such a way that as much of the pipe as possible fall under as low a type number as possible. The motivations are: (i) the execution time improves drastically from a higher order type to a lower order type, ar (ii) noise is reduced. The Phase II algorithms are such that the execution time is a direct function of the extent of the pipe segment, and the type number. Hence, even though the number of segments are increased and hence the overhead in computation, this overhead is offset and performance much betterd by following the said criterion. In short,

$$\text{execution time} \propto \sum_{i=1,n} (\text{type number of segment } i) \times (\text{extent of segment } i)$$

if there are n segments.

(by extent of the segment is meant the frame area required to accomodate the segment).

This criterion is demonstrated in Figure 5.3, Case B. The sumation on right hand side of the said relation can be much reduced by adopting a much finer segmentation, consisting of segments 12, 23, 34, 45, 56, 67, 78, due to the reduction in the total extent. However, the increased

number of segments produces a overhead in computation and input/output and hence maximum segmentation is not attempted. In general, whenever the pipe bend following a straight portion has the same type number, it is annexed to the straight portion. The segmentation in Figure 5.3, Case B follows this principle.

In the figure, the frames shown for each segment are only approximate. In reality, the frames are slightly larger, due to the following reasons:

(1) The frames should also contain the full diameter of the pipe, not just its centre line (as shown in figure). As a general procedure, the corners of the frame are extended outward by a distance = pipe radius. For a straight piece of pipe, extent = length of pipe x diameter of pipe.

(2) For each type, the frames are oriented and synchronized in characteristic ways, for considerations of speed (as mentioned earlier). This slightly enlarges the frames.

The complete Phase I algorithm for pipes is discussed in more detail in the following sections.

5.5 The Procedure for Pipes

Step 1: The points of the pipe are read sequentially. The reading of the pipe details, segmentation, pattern construction and writing on AUX2 (Phase I output) go simultaneously, point by point, from the starting point. At any instant, the complete details about four significant points are remembered. In the start, the first four points,

including starting point, are read in, and further points are read in when execution for the first point(s) have been completed. At times there may be less than four points, as when there are only two or three significant points for the pipe, or when only the last two or three points remain to be considered.

Step 2: Let the four points be 1,2,3 and 4, given by (x_1,y_1,z_1), (x_2,y_2,z_2), (x_3,y_3,z_3), (x_4,y_4,z_4), as in Figure 5.4. From the coordinates of these points, the angles of the bends at points 2 and 3 are calculated, given by ϕ_2 and ϕ_3. Figure 5.5 shows the relation that gives these angles, taking the pipe to occupy two sides of the triangle and the relation for 'offset' at the pipe bends, defined as the distance between the imaginary point of bending (the corner of the triangle) and the point where the pipe starts bending. Using this relation, the values 'offset 2' and offset 3' (Figure 5.4) are computed, at the points 2 and 3 respectively.

At this stage, it is checked that

$$\text{Distance } 1\ 2 \geq \text{offset}_2$$

$$\text{Distance } 2\ 3 \geq \text{offset}_2 + \text{offset}_3.$$

If they hold, the x,y,z coordinates of points B,C,D are computed.

Step 3: Pipe segment ABCD evidently lies in a plane. The plane passing through these four points of the form $aX + bY + cZ = d$ is determined (Section 5.6), and also the angles θ and θ_x (figures 5.6, 5.7). From these, the type of the frame that encloses the segment is found.

If the type of this segment ABCD is 1 or 2, execution proceeds with step 4, for the segment. If the type is 3 or 4, further segmentation is attempted, as follows:

(i) The plane and type number of segment AB is computed as before. If this type number is less than the type number of segment ABCD, then AB is considered as a separate segment.

(ii) The same procedure as (i) above is repeated for segment CD.

(iii) The type number of segment BC cannot be improved obviously. This, alongwith what remains of segments AB and CD [if (i) or (ii) fail] are clubbed together to form a segment, of the same type number as segment ABCD.

Thus step 3 results in finer segmentation; the portion ABCD may be split into atmost 3 segments (possibly 1 or 2). For each of these segments, step 4 onwards is carried out.

Step 4: This step consists of computing the 'link data' for each segment, and differs for each type. The segment at this stage consists of two, three or four significant points (the result of step 3); and may or may not enclose a curvature of given radius. To generlize, the segment is considered to compose of four points (A,B,C,D) always. Any adjacent two or three points may however coincide. Point T is the imaginary bending point (given in input).

The transformation of these points from the three dimensional coordinates to plane coordinates also takes place in this step. The plane coordinates are the coordinates of these four points along the plane of the segment (centre line plane), with respect to an origin. The origin, size and disposition of the frames are chosen to enclose the whole pipe segment and satisfying other considerations for bits synchronization.

In the discussions to follow, the representations used are:

(x_i, y_i, z_i) for the three dimensional coordinates of any point i.

(Px_i, Py_i) for the plane coordinates of any point i.

(CLX_o, CLY_o, CLZ_o) for the x,y,z coordinates of the origin of the centre line frame of the segment.

R = Radius of Curvature of the pipe, between points B,C

D_1, D_2= Pipe diameters for portions AB and CD respectively.

CRVRAD= Pipe [illegible] at the curve BC, = greater of (D_1, D_2)

Function AMAXOF $(P_1, P_2, P_3, \ldots)$ = Maximum of $P_1, P_2, P_3 \ldots$

Function AMINOF $(P_1, P_2, P_3, \ldots)$ = Minimum of $P_1, P_2, P_3, \ldots$

TYPE 4: NYPE = 4, oblique frames.

The link data for type 4 contains more information than other types. They include the following:

THETA, THETAX are the angles θ and θ_x, in radians

SINT, COST, [illegible] are the SIN, COS and TAN of angle

SINTX, COSTX, TANTX are the SIN, COS and TAN of angle θ_x

```
C*************************************************************
C                    FUNCTION ORG(N)
C*************************************************************
C
      ORG(N)=FLOAT(N-1)*PITCH
```

LISTING 5.1

```
C*************************************************************************
C     COMPUTATION OF LINK DATA FOR PIPES
C*************************************************************************
C
C     TYPE 1
      NTYPE=1
      NZLOW=NBIT(ZA-CRVRAD)
C     NOTE..CRVRAD=AMAXOF(D1,D2)/2.0 ,ZA=ZT=ZD
      NZHIGH=NBIT(ZA+CRVRAD)
      TEMP=AMINOF(XA,XT,XD)-CRVRAD
      NWX0=NBIT(TEMP)/36+1
      NY0=NBIT(AMINOF(YA,YT,YD)-CRVRAD)
      CLX0=ORG((NWX0-1)*36+1)
      CLY0=ORG(NY0)
      PXA=XA-CLX0
      PXT=XT-CLX0
      PXD=XD-CLX0
      PYA=YA-CLY0
      PYT=YT-CLY0
      PYD=YD-CLY0
      RETURN
C
C     TYPE 2
      NTYPE=2
      NZLOW=NBIT(AMINOF(ZA,ZT,ZD)-CRVRAD)
      NZHIGH=NBIT(AMAXOF(ZA,ZT,ZD)+CRVRAD)
      TEMP=AMINOF(XA,XT,XD)-CRVRAD
      NWX0=NBIT(TEMP)/36+1
      NX0=(NWX0-1)*36+1
      NYLOW=NBIT(YA-CRVRAD)
C     NOTE..YA=YT=YD
      NYHIGH=NBIT(YA+CRVRAD)
      CLX0=ORG(NX0)
      CLZ0=ORG(NZLOW)
      PXA=XA-CLX0
      PXT=XT-CLX0
      PXD=XD-CLX0
      PYA=ZA-CLZ0
      PYT=ZT-CLZ0
      PYD=ZD-CLZ0
```

```
C
C       TYPE 3
        NTYPE=3
        NZLOW=NBIT(AMINOF(ZA,ZT,ZD)-CRVRAD)
        NZHIGH=NBIT(AMAXOF(ZA,ZT,ZD)+CRVRAD)
        NXLOW=NBIT(XA-CRVRAD)
C       NOTE..XA=XT=XD
        NXHIGH=NBIT(XA+CRVRAD)
        NYD=NBIT(AMINOF(YA,YT,YD)-CRVRAD)
        CLYD=ORG(NYD)
        CLZD=ORG(NZLOW)
        PXA=YA-CLYD
        PXT=XT-CLYD
        PXD=YD-CLYD
        PYA=ZA-CLZD
        PYT=ZT-CLZD
        PYD=ZD-CLZD
        RETURN
```

LISTING 5.2

```
C*************************************************************

C       LINK DATA FOR TYPE 4 PIPES
C*************************************************************

        NTYPE=4
        DELT=PITCH*SINT
C       ALL TERMS ARE EXPLAINED IN FIGURE 5.12
        DXD=DELT*SINTX
        DYD=DELT*COSTX
        DZD=PITCH*COST
        F1=FLOAT(N1)+0.5
        FF=F1*DZD
        F3=CLZD+FF
        F4=CLZD-FF
        F2=(AMAXOF(PYA,PYT,PYD)+CRVRAD)*SINT
        IF(THETAX)700,702,702
C       ABS(THETA,THETAX)  ALWAYS LESS THAN 90 DEGREES
  700   F4=F4+F2
        GO TO 701
  702   F3=F3+F2
  701   NZHIGH=NBIT(F3)
        NZLOW=NBIT(F4)
        XOH=CLXD+ABS(F1*DXD)*ABS(THETAX)/THETAX
        YOH=CLYD+ABS(F1*DYD)
```

Initially the centre line frame origin, given by (CLXo, CLYO, CLZO), is taken as the point of intersection between (1) the line of intersection of the centre line frame with the horizontal plane at Z=0, and (2) the normal through (0,0,0) to this line (See Figure 5.6). With this point taken as the tentative origin, the values of PYA, PYT, PYB (the PY coordinates of the three points, in the pipe centre line plane) are computed as in Figure 5.7, and the values of PXA, PXT, PXB (the PX coordinates) as in Figure 5.8.

The extent of the frame thus obtained may not be of the minimal size. In fact, some of the PX_i, PY_i may even be negative. To make the frame correspond to the standard format, a series of origin shifts is performed. Whenever the origin is shifted by an amount (DPX, DPY), the new position of the origin in the three dimentional coordinates = (CLXO', CLYO', CLZO'), and the new positions of the three points in their own plane = (PXi', PYi') are to be recomputed The computations are shown in Figure 5.9. The following sequence of shifting is followed:

(i) Shift origin to point A; DPX = PXA, DPY = PYA

(ii) Test if point T is within the positive quarter of the frame. If it is within, proceed to Step (iii), else shift origin by DPX, DPY given as

DPX = PXT, if PXT < 0.0; else=0.0

DPY = PYT, if PYT < 0.0; else = 0.0

Points A and T are now within the positive quarter of the frame.

(iii) Repeat step (ii) for point D, so that all the points A,B,T,C,D are now within the positive quarter of the frame.

(iv) In order that the whole pipe diameter will also lie completely within the positive frame, shift the origin again by DPX = -CRVDIA/2.0, and DPY = -CRVDIA/2.0. The values of (CLXO, CLYO, CLZO) and (PXi, PYi) now, are their final values.

As a special case, when $\theta = 90°$, the rows of the frames are synchronized with the standard horizontal bit planes, by fixing the Z coordinate of the origin to a bit in the standard space. This is done as follows:

NZO = NBIT (CLZO)

CLZO = FLOAT (N20-1) * PITCH

PYi = Zi - CLZO, i = A,T,D

The following quantities are also computed, for use in remaining Phase I and in Phase II.

N1 = CRVDIA/(2.0*PITCH), trunctated to an integer

NN = N1 + 1 = number of frames to be constructed

N = 2 * N1 + 1 = total number of frames (See Fig 5.1

A total of N parallel frames would be necessary to completely contain the pipe. The pattern alongwith each of these frames is exactly similar, but for the width W_i of the pipe longitudinal sections. The frame numbered NN passes through the pipe centre line and forms the said centre line frame. Since the frames

placed symmetrically on either side of this centre line frame would contain exactly the same pattern, it is enough that only one set of frames are constructed so that NN frames are enough to define the pipe.

NOTE: Only in type 4, there is essentially, an actual frame (frame NN) passing through the pipe centre line. In types 1,2 and 3, the frames are constructed only along the horizontal and vertical standard bit planes and none of these need pass through the pipe centre line plane. They are however parallel to the pipe centre line plane (See Figure 5.11).

Step 5: Having filled up the link data, and determined the coordinates of the points in the pipe plane, the construction of patterns along the parallel planes is common for all the types; only the selection of the location of each frame with respect to the pipe C_L plane varies. This phase is discussed in detail in Section 5.7 alongwith the writing-on-tape procedures.

Step 6: The above procedures are executed for all the segments within the portion ABCD. Now the three points 0,3,4 remain. The following switching is performed:

point 1	←	point D
point 2	←	point 3
point 3	←	point 4
D1	←	D2
D2	←	D3

D3 ← D4

R2 ← R3

R3 ← R4

The new point 4 is read from the next card of the subsection (if any). Otherwise, the three points on hand only are considered. Execution repeats for the new set of points from step 2.

5.6 The Plane of the Pipe

Given the three (or two) points that define the segment, the plane of the pipe is first determined in the form ax + by + cz = d, from which the angles θ and θ_x are determined. When the number of points is less than 3(=2), and the plane is not unique, attempt is made to fit the lowest type of plane possible. The procedure is as follows:

[If there are only two points, assume (X3,Y3,Z3) = (X2,Y2,Z2)]

(i) If Z1 = Z2 = Z3, NTYPE = 1, a=b=0, d=1, c=1/Z1, retu
else(ii) If Y1=Y2=Y3, NTYPE=2, a=c=0, d=1, b=1/Y1, return
else(iii) If X1=X2=X3, NTYPE =3,b=c=0, d=1, a=1/X1, return
else(iv) NTYPE = 4

Let XD1 = X1, XD2 = X2, XD3 = X3. The listing attached gives the computation of the plane parameters.

5.7 Pattern Generation for Pipes

A part of the link data is computed in this phase also. They are:

LISTING 5.3

```
C*************************************************************
C                  PLANE THROUGH THREE POINTS (TYPE 4 ONLY)
C*************************************************************
C      POINTS ARE (X1,Y1,Z1),(X2,Y2,Z2),(X3,Y3,Z3)
C      PLANE EQN. IS  A*X+B*Y+C*Z=D, D=0.0 OR 1.0
       D=1.0
       XD1=X1
       XD2=X2
       XD3=X3
     1 P=(Z2-Z3)*(Y1-Y3)+(Z1-Z3)*(Y3-Y2))/
      1     ((XD3-XD2)*(Y1-Y3)+(XD3-XD1)*(Y3-Y2))
       Q=((Z2-Z1)*(XD2-XD3)+(Z3-Z2)*(XD2-XD1))/
      1     ((Y1-Y2)*(XD2-XD3)+(Y2-Y3)*(XD2-XD1))
       R=P*XD1+Q*Y1+Z1
       IF(R.GT.0.00005)GO TO 2
C      THE PLANE PASSES THROUGH ORIGIN., D=0.0
       D=0.0
C      SHIFT THE ORIGIN, FIND A,B,C -THEY DO NOT CHANGE
       XD1=XD1+1.0
       XD2=XD2+1.0
       XD3=XD3+1.0
       GO TO 1
C
     2 C=1.0/R
       A=P*C
       B=Q*C
       THETAX=ATAN(A/B)
       THETA=ATAN(A/(C*SIN(THETAX)))
C      ABS(THETA,THETAX) ALWAYS LESS THAN 90 DEGREES
C      DEDUCTION OF THE ABOVE RESULTS IS STRAIGHT FORWARD, AND
C       IS NOT DISCUSSED HERE
       RETURN
```

NP = NBIT [AMAXOF(PX1,PXT,PXD)+CRVDIA/2.0]

NWPX = NP/36+1

NPX = NWPX*36

NPY = NBIT [AMAXOF(PYA,PYT,PYD) + CRVDIA/2.

NWORDS = NWPX * NPY

NPART = 5000/NWORDS (truncated)

The record size = 5000 words, is fixed, and hence NPART, for a given NWORDS. If this number of partitions is sufficient to accomodate the required number of frames, there will be only one record. Else, the remaining frames are accomodated in the next record(s) with suitable modifications in the link data. It is ensured that each of these records can be considered independently in Phase II.

If NTYPE = 2 or 3, NZHIGH = NZLOW + NPY - 1

If NPART > required number of partitions,
NPART ← required number of partions.

The required number of partitions = NZHIGH - NZLOW + 1 for type 1

= NYHIGH-NYLOW + 1 for type 2

= NXHIGH-NXLOW + 1 for type 3

= NN for type 4.

The patterns are constructed frame by frame. For types 1,2 and 3, frames are selected along the standard horizontal or vertical bit planes (as the case may be). For type 4, one frame coincides with the centre line frame and the rest of the frames are parallel to this frame. The distance between two frames is always = pitch.

For each frame, the offset of the frame from the centre line of the pipe is determined (Figure 5.13). The bit pattern of the pipe segment ABCD along this frame is constructed in three stages (See Figure 5.14):

(i) The section AB of the pipe segment is first inserted. It is a rectangle from point (PXA,PYA) to (PXB,PYB), with a width W_i depending upon the offset of the frame from pipe $\mathcal{L}$ and the pipe diameter D_1 (W_i computed as in Figure 5.13).

Let ϕ_1 = TAN^{-1}[(PYB-PYA)/(PXB-PXA)], in radians, in the range 0 to 2π

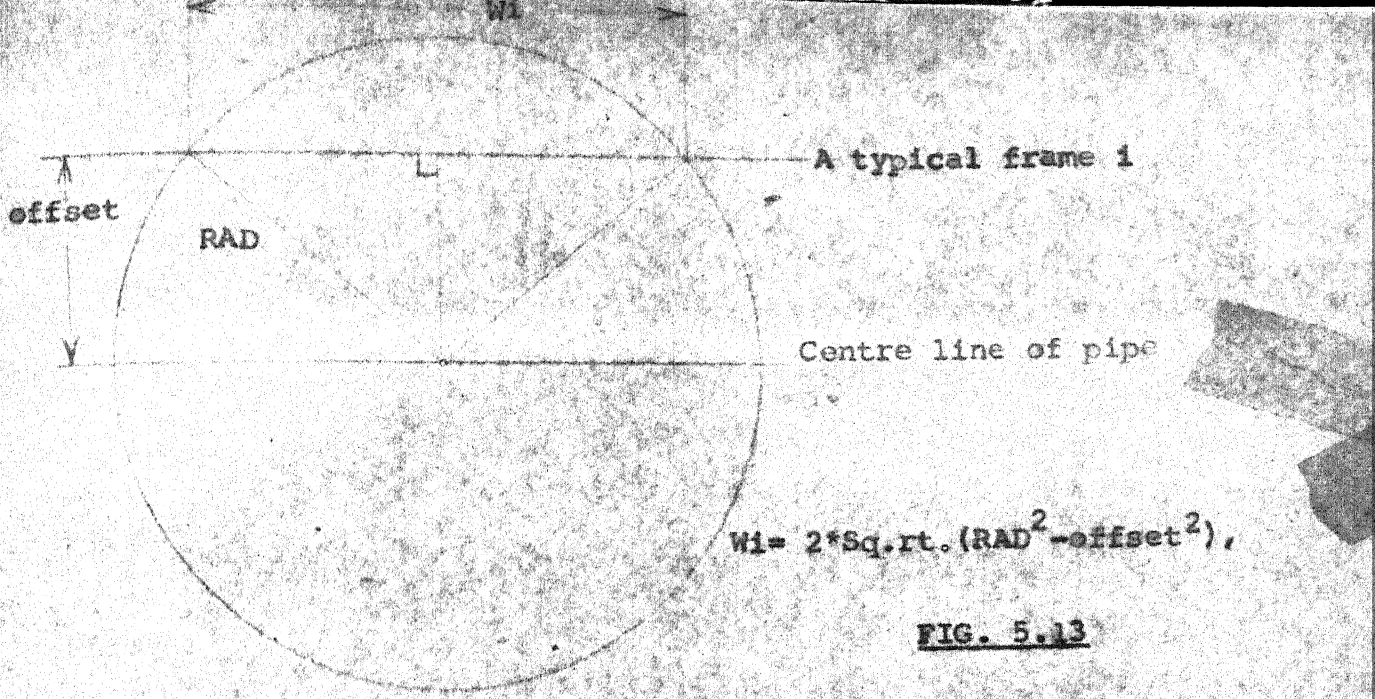
= slope of the axis AB of the rectangle

(ii) The section CD is inserted next. The width is computed similarly from the offset and the pipe diameter D_2 at this section, and the bit pattern for the rectangle is inserted in the frame.

Let ϕ_2 = slope of the axis CD of rectangle, in radians, in the range 0 to 2π

Note: ϕ_1 and ϕ_2 are computed only once and not for every frame.

(iii) The curve BC remains. The width of the portion BC is the greater of the widths of the portions AB and CD. The curve BC is considered in small rectangles, that together approximate the curvature. Each rectangle subtends an angle



$Wi = 2 * Sq.rt.(RAD^2 - offset^2)$,

FIG. 5.13

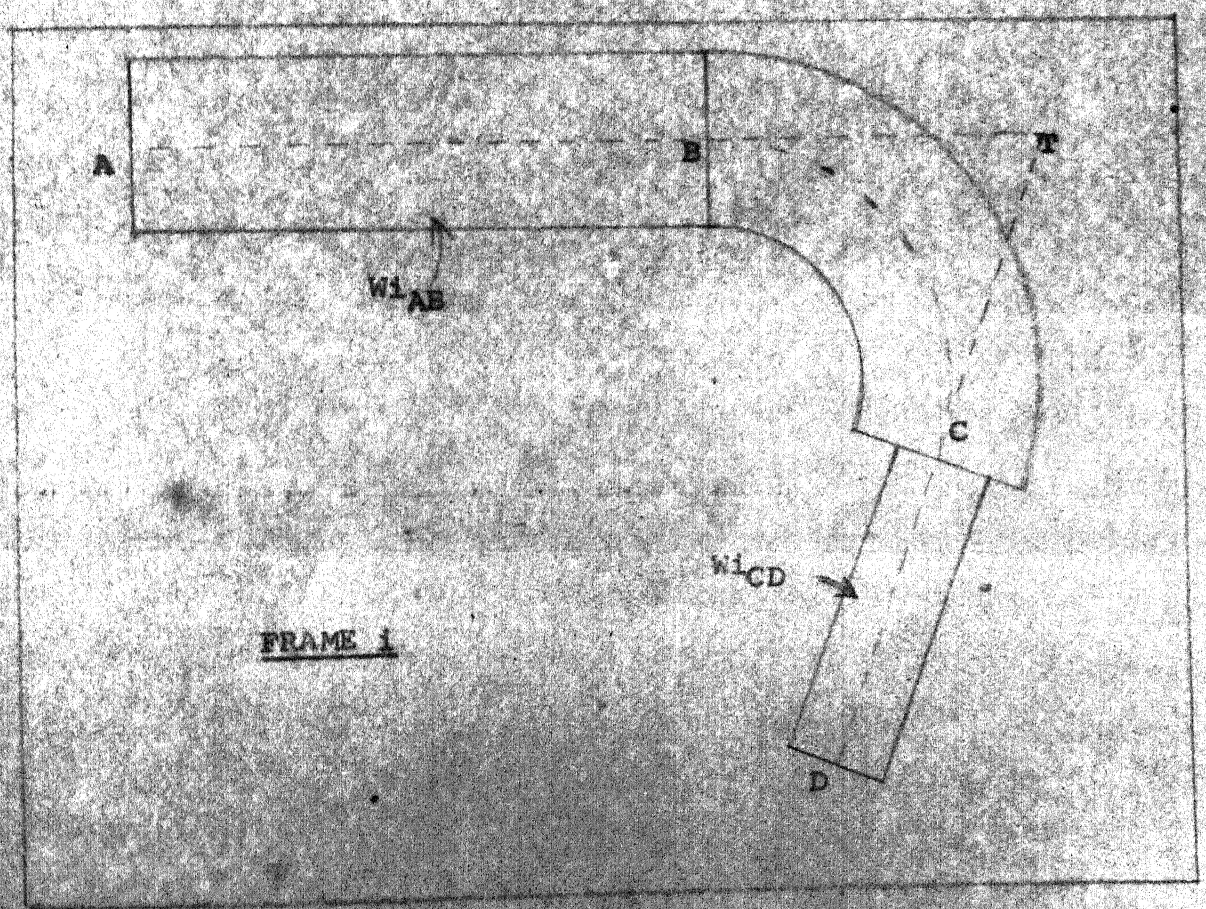
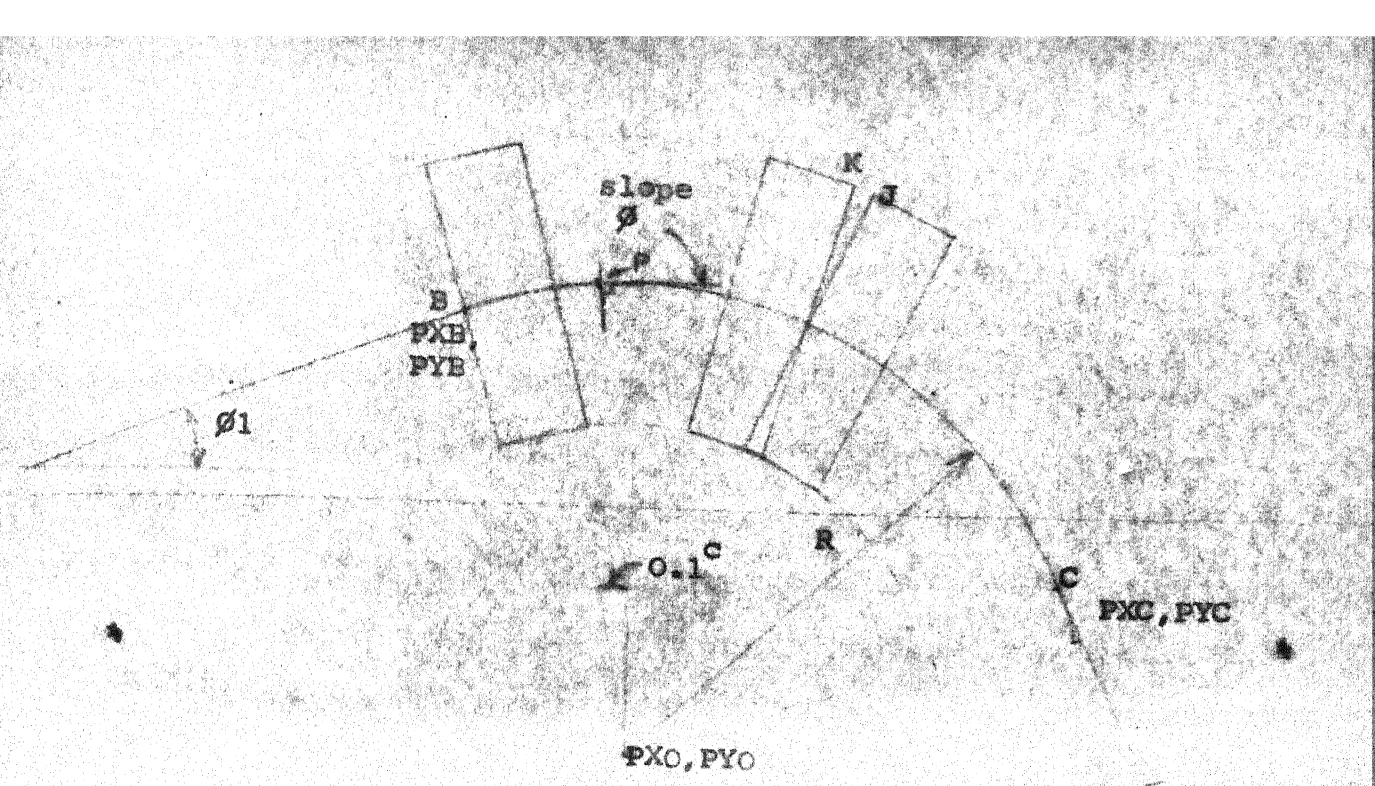


FIG. 5.14

of 0.1 radians at the centre of curvature (PX_o, PY_o); and the ends of the rectangles are computed as in Figure 5.15. The slope of the first rectangele is ϕ_1+0.05 radians, and those of subsequent rectangles are obtained by incrementing it by 0.1 radians each time. The rectangles are constructed till the slope ϕ of such a rectangle just exceeds the slope ϕ_2, at which stage the curve BC would have been completely covered.

This splitting up of the arc into rectangles may cause a rupture (KJ in Figure 5.15) at the outer edge of the arc. If the distance KJ exceeds the value of pitch, problems might arise. It can be shown that KJ = (Pipe diameter *f)/2.0, where f = angle subtended at the centre of curvature, by each rectangle. If KJ should always be less than pitch, it can be derived that f < (2.*PITCH)/Pipe diameter. Whenever the pipe diameter is too large, and this relation does not hold for f=0.1 radians, f is suitably decreased.((Another solution for this problem is to overlap the individual rectangles. Since this is only a worse approximation, and may not appreciably reduce the number of rectangles, it is not adopted).

Each of these rectangles is constructed in a scratch frame of the same size, and then 'or'ed into the required frame.



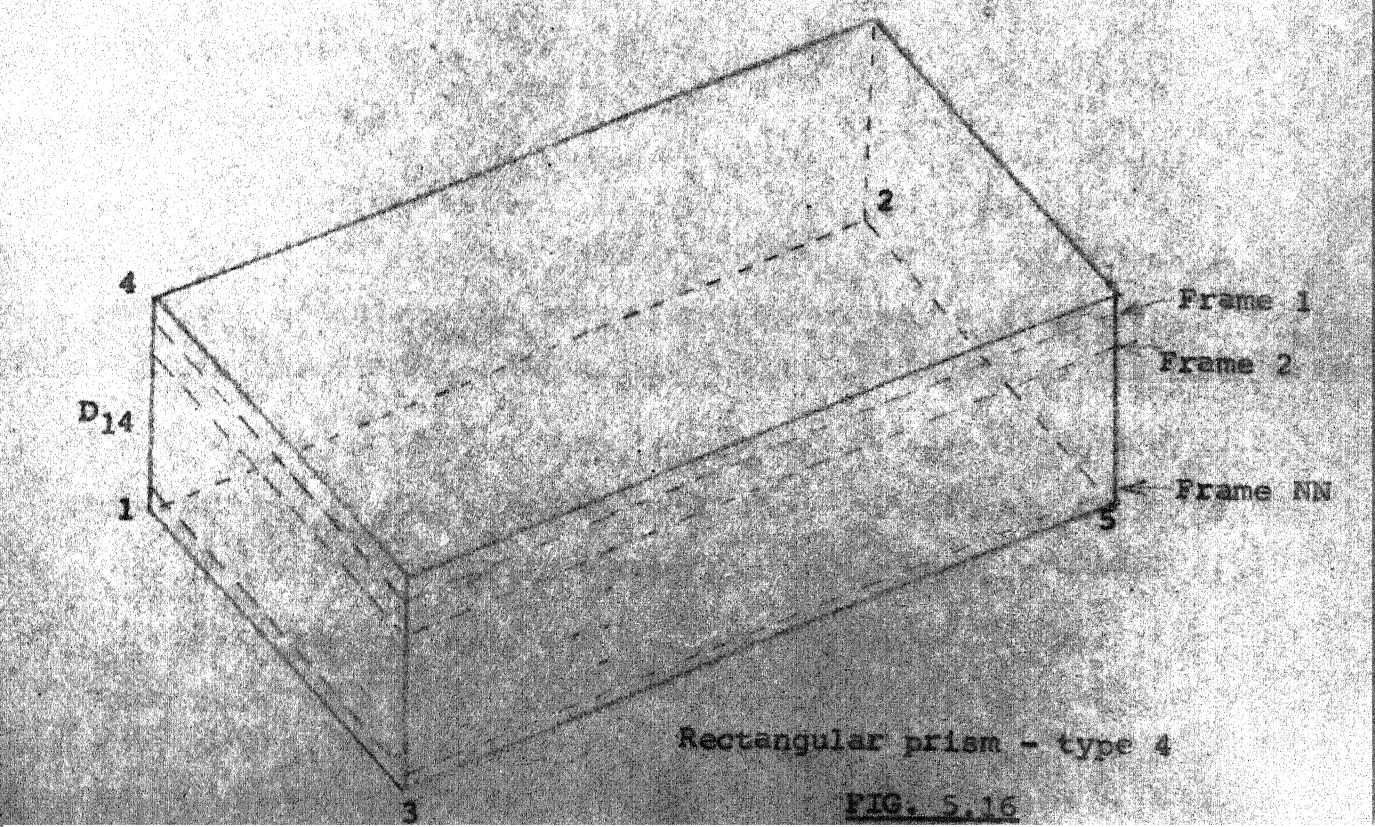
$PXO = PXB + R*SIN(\emptyset_1)$

$PYO = PYB - R*COS(\emptyset_1)$

FIG. 5.15

For any point P, PXP = PXO - R*SIN(Ø)
PYP = PYO + R*COS(Ø)

(The actual relations will have to take into account the direction of bend, etc., and will be more complex)



Rectangular prism - type 4

FIG. 5.16

When all the required frames have been completed, or when the record (of 5000 words) cannot take any more frames, the record with the link data are written out. In the latter case, all the relevant data are suitably modified. For example - in type 1, NZLOW, NZHIGH; in type 2, NYLOW, NYHIGH; in type 3, NXLOW, NXHIGH and in type 4, NLOW, NHIGH (the lower and the upper frame numbers in the current record). In type 4 if all the NN frames (minimum required) go into the same record, there is no problem. However, if NPART < NN, the first NPART records are built into the first record. Now, to make the second record entirely independent of the first record (to generalize Phase II procedures), all the N frames are constructed, instead of the minimum requirement of first NN frames. Thus, if a type 4 segment does not go into a single record, it takes at least 3 records. Since it is ensured that type 4 segments are of minimum size, they normally go into one record.

5.8 Non-standard Shapes

The less common and non -standard shapes in a plant, as valves, boilers, etc. are defined by composing them from primitives. The primitives available are rectangular prisms, triangular prisms, cylinders, spheres, half cylinders and hemispheres. The permitted composing operations are OR, ADD (both are essentially the same), AND and REMOVE. No checking is performed while adding or removing of primitives during composition. The engineer can imagine an initially empty

space to/from which primitives of given size and orientation are added/'and'ed/removed. Hence the sequence of composing operations is extremely important. Interchanging them might entirely change the resultant shapes. By nature of the procedure adopted, it is not possible to compose two or more shapes separately and operate on them.

Each primitive shape is defined by its significant points. The most significant point (or one of the points) is defined in space in absolute x,y,z coordinates, and the rest of the points, if required, are defined with respect to this point, in terms of the differences in x,y, and z to reach the new point from the first point. These differences, denoted by dx, dy, dz, are positive in the positive direction of x,y,z respectively. Each card of the data section (or a set of two adjacent cards) defines a primitive completely. For each primitive read in, after the initial format check, screening and data printout operations, the plane of the primitive is computed, the bit patterns are constructed along parallel planes, and the record(s) output on AUX2. Thus, there will be atleast as many records written out as the number of composing primitives, in the section. As for pipes, attempt is made to fit in each primitive into the lowest type of number possible. The actual composing operations upon these primitives is done only in Phase II while extracting the horizontal sections; the operation to be performed

on each primitive is stored on tape alongwith the link data.

5.9 Rectangular Prisms

The fact that the length, and the two sides of a rectangular prisms can be interchanged with simple modifications on their definitions, is utilised to fit these primitives into the lowest type number. The orientation (and hence the type number) of each of the three orthogonal faces of the prism are computed, and it is fitted to the lowest type number obtained.

From the input data, the x,y,z coordinates of four corners of the prism can be easily obtained. It is checked that these four points do not occupy the same plane, which indicates an error in data preparation. This check is done by fitting the plane along three points and testing if the fourth point also lies in the same plane.

If the plane obtained above is of type 1, no improvement is possible. If the type is higher, planes are fitted through the other two combinations of three corners each. The particular combination giving the lowest type number is taken. (Refer 5.6 for fitting the planes).

Let the three points corresponding to the lowest type number be points 1, 2 and 3, and the fourth be point 4 (given by x_i, y_i, z_i, $i = 1,4$). Since the actual frames along which patterns are to be constructed are all parallel to one of the faces of the prism, the pattern along each frame

is exactly the same, a rectangle, whose three corners are defined by the points 1,2 and 3. Hence, only one frame need be actually constructed. The link data and the coordinates (PX_i, PY_i) of these three points in their plane are computed as in the listing. The point 5(X5,Y5,Z5), which completes the rectangle alongwith points 1,2,3, is computed as:

$$X5 = X2 + (X3 - X1)$$

$$Y5 = Y2 + (Y3 - Y1)$$

$$Z5 = Z2 + (Z3 - Z1)$$

In type 4

PX1, PX2, PX3, PX5 are computed as in Figure 5.8. By shifting the origin suitably, the points, 1,2,3,5 are brought into the positive quadrant of the frame. The procedure adopted is:

(i) Shift origin to point 1, by having

DPX = PX1, DPY = PX1 (refer. 5.9 for shifting)

(ii) If point 2 is not in positive quadrant, shift again with

DPX = PX2, if PX2 < 0.0; else 0.0

DPY = PY2, if PY2 < 0.0; else 0.0

This brings point 2 within positive quadrant.

(iii) Repeat step (ii) for points 3 and 5.

LISTING 5.4

```
C*************************************************************
C     LINK DATA CALCULATION FOR RECTANGULAR PRISMS
C*************************************************************
C     TYPE 1
      NTYPE=1
      NWX0=NBIT(AMINOF(X1,X2,X3,X5))/36+1
      NX0=NWX0*36+1
      NY0=NBIT(AMINOF(Y1,Y2,Y3,Y5))
      NZLOW=NBIT(AMINOF(Z1,Z4))
C     NOTE..Z1=Z2=Z3
      NZHIGH=NBIT(AMAXOF(Z1,Z4))
      CLX0=ORG(NX0)
      CLY0=ORG(NY0)
C     COMPUTATION OF PX,PY FOR POINTS 1,2,3,5 SIMILAR TO PIPES
      RETURN
C     TYPE 2
      NTYPE=2
      NWX0=NBIT(AMINOF(X1,X2,X3,X5))/36+1
      NX0=NWX0*36+1
      CLX0=ORG(NX0)
      NZLOW=NBIT(AMINOF(Z1,Z2,Z3,Z5))
C     NOTE..Z4=Z1 OR Z2 OR Z3
      NZHIGH=NBIT(AMAXOF(Z1,Z2,Z3,Z5))
      CLZ0=ORG(NZLOW)
      NYLOW=NBIT(AMINOF(Y1,Y4))
      NYHIGH=NBIT(AMAXOF(Y1,Y4))
      RETURN
C     TYPE  3
      NTYPE=3
      NY0=NBIT(AMINOF(Y1,Y2,Y3,Y5))
      CLY0=ORG(NY0)
      NZLOW=NBIT(AMINOF(Z1,Z2,Z3,Z5))
      NZHIGH=NBIT(AMAXOF(Z1,Z2,Z3,Z5))
      CLZ0=ORG(NZLOW)
      NXLOW=NBIT(AMINOF(X1,X4))
C     NOTE..X1=X2=X3
      NXHIGH=NBIT(AMAXOF(X1,X4))
      RETURN
C     TYPE 4
      NTYPE=4
C     A,B,C,D ARE THE PLANE PARAMETERS
      D14=SQRT((X4-X1)**2+(Y4-Y1)**2+(Z4-Z1)**2)
      N=NBIT(D14)
      NN=N
      NLOW=1
      NHIGH=N
C     REFER FIGURE 5.16
C     INITIAL CLX0,CLY0,CLZ0 COMPUTED AS IN FIG. 5.6,5.7
      CLZ0=0.0
      CLX0=D*A/(A**2+B**2)
      CLY0=D*B/(A**2+B**2)
```

Now all the points 1,2,3,5 are in the positive quadrant; the new (PX_i, PY_i) give the four points in their frame, and (CLXO, CLYO, CLZO) now gives the origin of this frame.

Figure 5.16 shows the convention following in numbering the frames. If Z4 $\geq$ Z1, the plane of points 1,2,3,5 forms frame NN, and its origin is defined by (CLXO, CLYO, CLZO). No change is hence required. If however Z4 < Z1, the plane of points 4,6,7,8 form frame NN, and the origin computed above is the origin of frame 1. The convention requires that (CLXO, CLYO, CLZO) should be the origin of frame NN, and hence the origin is corrected as:

$$\begin{aligned} CLXO &= CLXO + (X4 - X1) \\ CLYO &= CLYO + (Y4 - Y1) \\ CLZO &= CLZO + (Z4 - Z1) \end{aligned} \qquad 5.1$$

However, the (PX_i, PY_i) are just the same, and need not be corrected. Further, if $\theta = 90°$, the origin is synchronized with a standard horizontal bit plane, as discussed in Section 5.2. The other link data are computed exactly the same way as for pipes type 4.

Pattern Construction: After the link data have been computed, the pattern construction is common for all the four types. As mentioned, only one frame need be actually constructed. A rectangle is inserted, with an axis AB (Figure 5.17) and width W, computed as in the figure. In the process, the following common link data are computed:

NWPX = NBIT [AMAXOF(PX1,PX2,PX3,PX5)]/36 + 1

NPX = NWPX * 36

NPY = NBIT [AMAXOF (PY1, PY2, PY3, PY5)]

NWORDS = NWPX * NPY

NPART = 1 (only one frame is actually constructed, and repeated for the NN frames)

5.10 Triangular Prism

The only difference in the procedures for the rectangular and triangular prisms is that the bit patterns along the parallel frames are not the same. As in the case of rectangular prisms, the type number of the three longitudinal faces (not the end faces) are determined, and the frames are constructed parallel to that face for which the type number is minimum. (See Figure 5.18.). Only the differences in procedure are mentioned below. The other steps are the same as in 5.9.

The faces considered in rectangular prism are CAB, CAD, DAB (Figure 5.19) whereas in triangular prism, the faces CAB, BAD and EDC (Figure 5.20) are considered. The point E is computed from the coordinates of points A,B and D, as ABED forms a rectangle. Let points 1,2,3 form the face with minimum type number.

The number of frames required (NN) is computed from the distance D4P (Figure 5.21), rather than the distance D14 as in the case of rectangular prisms. The point P, and hence D4P, can be computed from the coordinates of points 1,2 and 4.

In case the frames are of type 4, and if Z4 < ZP, the frame numberings have to be changed as in case of rectangles and hence the origin is to be shifted too to that of frame NN (See Figure 5.22). The change is effected by the vector $\overline{P4}$ rather than the vector $\overline{I4}$ (as in rectangles). Hence the equations 5.1 become

$$CLXO = CLXO + (X4 - XP)$$

$$CLYO = CLYO + (Y4 - YP)$$

$$CLZO = CLZO + (Z4 - ZP)$$

The pattern construction phase differs from that of the rectangular prism. Along each of the NN frames, the pattern is essentially similar, but for the width W_i and the position of the axis of the rectangle, $0_i0_i'$ (Figure 5.23). The rectangles defined by $0_i,0_i'$ and W_i are inserted into corresponding frames, starting from frame 1.

5.11 Cylinders

A cylinder is treated exactly the same was a straight segment of pipe. The cylinder definition is taken for a full subsection of a pipe. The conversion of the input data to suitable form is done internally.

5.12 Semi-Cylinders

The semi cylinder in effect is a straight pipe segment, for which only frames 1 to NN, or frames NN to N are to be considered (depending upon which half of the cylinder is

present. There is not much freedom in fixing the plane and hence type number of the semi-cylinder; the type number is completely fixed by the centre line plane of the cylinder at which it has been cut to obtain the semi cylinder. This plane is computed first, alongwith θ, θ_x and type number. The numbering of the frames follows the same convention as for other objects, and the origin (CLXO, CLYO, CLZO), corresponds to the centre line frame (frame NN). From the input information, as to which of the halves is present, and the details of the plane, the program decides which of the frames (1 to NN or NN to N) are to be inserted. The values of NLOW and NHIGH are correspondingly fixed: (i) for frames 1 to NN, NLOW = 1 and NHIGH = NN. If NN frames do not go into a single record, let M frames be accomodated in the first record. Then NLOW = 1 and NHIGH = M for first record, and NLOW = M+1 and NHIGH = NN for the next record. This goes on till all NN frames have been accomodated. (ii) for frames NN to N, NLOW = NN and NHIGH = N. The rest of the procedure is same as for (i) above.

With this exception, the same procedure as for pipes is followed.

5.13 Spheres

A sphere is defined by its centre and radius. The shape is as convenient to handle in any type of plane, and type 1 is chosen. The patterns are constructed along parallel

horizontal frames, whose origins are synchronized to a bit in t he standard bit space, as for all type 1 frames. The pattern on each of these horizontal frames is a circle, with their centres falling in a vertical line. Only the radii of these circles, R_i, differ, depending upon the 'offset' of these frames from the centre of the sphere, and are calculated as in Figure 5.24.

The link and other relevant data (for type 1) are computed as in other cases:

Let centre be (XC, YC, ZC) and radius = R

```
NTYPE  = 1
NZLOW  = NBIT (ZC - R)
NZHIGH = NBIT (ZC + R)
NWXO   = NBIT (XC - R)/36 + 1
NYO    = NBIT (YC - R)
PXC    = XC - POINT (NWXO * 36 + 1))  Coordinates of
                                    )  the centre, in
PYC    = YC - POINT (NYO)           )  the frames
NWPX   = NBIT (PXC + R)/36 + 1
NPY    = NBIT (PYC + R)
NWORDS = NPY * NWPX
NPART  = 5000/NWORDS or number of frames required,
         (=NZHIGH-NZLOW+1), whichever is lower.
```

The frames are filled up from frame 1 (at NZLOW). If a single record does not suffice the NZLOW and NZHIGH of the present and next record(s) are suitably modified, and left over frames are built into the next record(s).

Pattern Generation: For any frame, the 'offset' is calculated as:

Offset = ABS (ZC - NZ * PITCH)

where

NZ = horizontal bit plane number (NZ varies from NZLOW to NZHIGH)

The radius of the circle in that frame ($=R_i$) is calculated from offset as in Figure 5.24. These circles are constructed in an exactly similar fashion - knowing the coordinate of the centre, the radius R_i of the circle and the offset of any bit row in the frame from that centre, the starting and ending point of the occupied space along that row are computed as in Figure 5.25. The bits in that row between these limits are turned on by an appropriate routine. This proceure is repeated for each of the NPY rows in the frame, to complete the circle.

The whole procedure is repeated for each frame.

5.14 HEMISPHERES

The procedure for a hemisphere is a hybrid of the procedures adopted for pipe segments and spheres. The plane which cuts the sphere to obtain the hemisphere is determined first from the input data. The type of the frames is entirely decided by the type of this plane, as the frames are constructed parallel to this plane.

Whatever the type number, the patterns in each of the frames are circles, with the same centre PX_c, PY_c, but with various diameters. The diameter of these circles depends upon the distance of the frame concerned from the sphere centre.

Type 1: The cutting plane is horizontal

If the lower hemisphere is present, NZLOW = NBIT (ZC-R)
NZHIGH = NBIT (ZC)

If the upper hemisphere is present, NZLOW = NBIT (ZC)
NZHIGH = NBIT (ZC+R)

The rest of the procedure exactly follows that for a sphere

Type 2: The cutting plane is vertical, parallel to X axis.

If the inner side hemisphere is present (i.e. the hemisphere is on the origin side of its centre)

NYLOW = NBIT (YC - R)

NHIGH = NBIT (YC)

If the outer side hemisphere is present, NYLOW = NBIT (YC)
NYHIGH = NBIT (YC+R)

For both NWXO = NBIT (XC-R)/36+1
NZLOW = NBIT (ZC-R)
NZHIGH = NBIT (ZC+R)
PXC = XC - [illegible] [(NWXO-1)*36+1]
PYC = ZC - [illegible] (NZLOW)
NWPX = NBIT (PXC+R)/36 + 1
NPY = NBIT (PYC + R)

The rest of the procedure is similar to that for spheres, with a few modifications for the change in type number.

Type 3 The cutting plane is vertical, parallel to Y axis

For inner hemisphere, NXLOW = NBIT (XC - R)
NXHIGH = NBIT (XC)

Four outer hemisphere NXLOW = NBIT (XC)
NXHIGH = NBIT (XC + R)

For either, NZLOW = NBIT (ZC - R)
NZHIGH = NBIT (ZC + R)
NYO = NBIT (YC - R)
PXC = YC - [illegible] (NYO)
PYC = ZC - [illegible] (NZLOW)
NWPX = NBIT (PXC+R)/36, rounded to next higher integer
NPX = NWPX * 36
NPY = NBIT (PYC + R)

The rest of the procedure is similar to that for spheres, with suitable modifications for the change in type number.

Type 4: The cutting plane is oblique.

As in the case of type 4 cylinders and pipes, the centre line frame, numbered frame NN, is essentially present [where NN = NBIT (R) = number of frames required]. As before N = 2 * NN - 1. With the standard numbering of frames, there are two possibilities:

(i) NLOW = 1, NHIGH = NN
(ii) NLOW = NN, NHIGH = N

as only half the sphere is to be put in. The case (i) or (ii) is chosen by the program, depending upon the input.

The origin [CLXO, CLYO, CLZO] of the centre line frame is computed on the same principles. The origin is tentatively assumed at a point, as in the case of pipes (type 4), and the (PX_C, PY_C) of the sphere centre are computed. It is then

shifted to the sphere centre, by having DPX = PXC, DPY = PYC. It is shifted again, by an amount DPX = -R, DPY = -R, so that the frame would accomodate the whole circle at the frame NN (with full dia.), in its positive quadrant. A further shift is made in the origin, if $\theta = 90°$, to synchronize the frame origin with a horizontal bit plane. After all these shifts, what remains of (CLXO, CLYO, CLZO) and (PXC, PYC) are their final values. The frame size is next computed as NWPX = NBIT (PXC + R)/36 rounded to next higher integer and NPY = NBIT (PYC + R).

The rest of the procedure is exactly same as for pipes, but for the pattern construction. Instead of combinations of rectangles, as in the pipes case, here, circles are constructed with centre at PX_c, PY_c, and diameter determined as explained earlier for spheres (from the offset of the frame from the centre of the sphere).

5.15 Object Predefinitions

The complete standardization of procedures for all objects has made possible an efficient system of predefining shapes that occur frequently in the plant. Instead of composing these shapes repeatedly, which is a waste of time both for the engineer and the computer, these shapes are composed just once, and their bit patterns are stored permanently in a convenient form. For the given origin and frame directions

of the shape (refer Figure 5.26), the parallel frames containing the bit pattern of the shape are stored in AUX1, file 2, (as Phase I output). These shapes can be fitted later into any other place, by suitably specifying the new origin, axial plane orientations, and scale. The link data are computed, from the distance between the two origins and the changes in reference plane (axial plane) orientations (θ and θ_x) and these patterns are reproduced with the new set of link data, as the Phase I output. Thus the major part of Phase I computation is avoided. When a scaling down is requested, the frames are reconstructed by a fairly fast collapsing procedure, which effectively increases the pitch to the normal value (as, due to scaling down, the pitch of the stored patterns get reduced). Scaling up is not attempted, as it is a matter of reducing the pitch, or, getting more information from the patterns than what is available. It may not be very reliable.

While predefining the shape, the procedure followed is exactly the same as for actual on the spot definition of non standard shapes (discussed so far) but for the fact that the records are written in a different portion of the tape. Alongwith the normal information, the code number assigned and the name of the shape and the origin, and orientations of the reference plane of the predefined shape are also stored. The type of these records may be anything, and are suitably altered to suit the new orientations. It is obvious that the

actual patterns need not be changed; only the details defining the plane of the object are to be changed.

CHAPTER VI

PHASE II EXECUTION

6.1 Outline

Phase II execution comprises of extracting the horizontal sections (bit patterns, synchronised with standard horizontal bit planes in all respects), from the Phase I output. Bit patterns are extracted corresponding to only those bit planes, which are affected by the current data section. As mentioned earlier, no discrimination is made in Phase II on the kind of object. All the objects are treated exactly alike.

The Phase I output consists of records of patterns, alongwith the link data that help interpret these patterns. Besides other details, the link data of each record has details on the minimum and maximum bit planes affected (NZLOW, NZHIGH) by that record, the type of the record (1 to 4) and the operation to be performed with that record (Add, OR, And or Remove). These details are used initially in branching to the appropriate procedure.

In Phase I, the total number of records output, the minimum horizontal bit plane affected by all these records (=MIN2) and the corresponding maximum (=MAX2) are also

computed. In Phase II, the extraction procedure starts with bit plane MIN2 and proceeds upto MAX2; these horizontal bit sections are extracted one by one from the Phase I output and stored in AUX1. sequentially.

For each bit plane, IZ (between MIN2 and MAX2), the Phase I output records are read sequentially from the first record, and patterns extracted and accumulated. When IZ is outside the range of a particular record i.e., IZ outside range NZLOW to NZHIGH), that record does not contribute any pattern to the bit plane IZ, and is hence skipped. Since the Phase I output records have been written in the exact input sequence, the composition sequence (for non standard shapes) are maintained by reading them back sequentially and operating on them.

The Phase II output consists of one frame per record, giving the net horizontal section of the new object to be added or removed (as given in the operation code in the section starting card) to/from the plant. The size of these frames is exactly the same as that of the standard horizontal bit planes, and cover the whole plant, even if the new object occupies only a small part of the whole frame. Thus, the memory and time utilisation improve as more objects are considered in a single section. Further, the Phase II output frames are completely synchronized in space with corresponding standard horizontal bit planes, bit to bit.

In contrast, the Phase I output, as discussed in the last chapter, may contain more than one frame per record (all parallel); the frames themselves are of any size and orientation.

The linking from Phase II to Phase III is maintained through the parameters NTOT and INDIM, and the table INDEX. NTOT gives the number of phase II output records (initialized to 0 and incremented each time), INDIM gives the dimension of the table INDEX, and the table INDEX has the range of bit planes affected, for each record.

If INDIM=0, INDEX is a single dimension array, with one entry per record. There are NTOT entries, and each entry gives IZ for the corresponding record on AUX1. (IZ = the horizontal bit plane number). Normal Phase II operations a re in this mode.

If INDIM=1, INDEX is a double dimensioned array, with two entries per record. There are 2 * NTOT entries. Each pair of entries gives NBOT and NTOP for the corresponding record on tape (NBOT = lower standard bit plane number affected, NTOP = the upper standard bit plane affected, by this record. NBOT $\leq$ NTOP). This mode is adopted only for columns, floors and beams, for which Phases I and II are merged. They are discussed separately at the end of this chapter.

For extracting each bit plane IZ, the Phase I output records are read sequentially into a 5000 word buffer. For

those records which affect the bit plane IZ, one of the standard extraction procedures is taken depending upon the type number of the record. The link data available alongwith the Phase I records are extensively utilised in the extraction. (The explanations of these data are available in Chapter V and are not repeated here). The extracted patterns from each record are sequentially superimposed on another 5000 word buffer which was first initialized to all zeroes. This buffer is latter written out and the procedure is repeated for the next IZ.

The extraction procedures are discussed in the following sections for each type of record.

6.2 TYPE 1

One of the frames in the record, corresponding to the level IZ, is to be superimposed on the accumulated (extracted) pattern. Since the PX side of the frame is a multiple of 36 (word length), the origin of the frame is synchronised to the first bit of a machine word in the standard bit plane, and the frame is synchronized with the standard horizontal bit plane bit to bit, the problem reduces to 'or'ing together (or 'And'ing or removing from, as the case may be) the corresponding machine words in the frame and the extraction buffer, a nd storing them back in the latter.

The starting address of the frame (within the record) is given as

M = (IZ - NZLOW) * NWORDS + 1, if NPART > 1

M = 1, if NPART = 1 (i.e., the same frame has to be repeated for all the levels from NZLOW to NZHIGH)

The address of the word N (See Figure 6.1) in NATRIX (extraction buffer) is given by

N = (NYO - 1) * NWX + NWXO

There are NPY rows in the frame NATRIX, each NWX word long. For the first row, the NWX words starting from word M in MATRIX (the Phase I record) are to be operated with the NWX words starting from word N in NATRIX, and the result stored back in the same position in NATRIX. For the second row of the frame, N is incremented by NWX, the plant length in words, and the same procedure is repeated.

6.3 TYPE 2

A type 2 record consists of a set of parallel vertical frames, each frame exactly synchronized, bit to bit, with standard vertical bit planes, parallel to X axis. Further, the origin of each frame is also fixed to the first bit of a machine word in standard bit space. Since each row of the frames consists of an integral number of machine words, the first word of each row is synchronized with a corresponding word in the bit space, and so also all the words that follow.

Each individual frame in the record has its row 1 at level NZLOW and row NPY at level NZHIGH, and the frames are synchronized in space with successive vertical bit planes from NYLOW (for frame 1) to NYHIGH (for frame NPART). (See Figure 6.2). Thus, when level IZ is being extracted, only row (IZ - NZLOW + 1) of each frame is to be considered (as it is synchronized with the level IZ). The corresponding machine word 'IPWORD', expressed with respect to the frame origin is

$$IPWORD = (IZ - NZLOW) * NWPX + 1$$

IPWORD gives the starting address of the particular row, within a frame.

The NWPX words of the first frame, starting from word IPWORD, are to be operated with the NWPX words of NATRIX, starting from word N, where N is given as

$$N = (NYLOW - 1) * NWX + NWXO$$

The result is stored back in the same place in NATRIX.

The corresponding row of the next frame is to be operated next with the next row in the standard horizontal bit space (NATRIX). For this, N is incremented by NWX (the plant length in words), and the new IPWORD becomes

IPWORD = IPWORD + NWORDS, if NPART > 1

IPWORD = IPWORD, if NPART = 1 (same frame repeats from NYLOW to NYHIGH)

This procedure is repeated for each NY from NYLOW to NYHIGH.

6.4 Type 3

The frames in a type 3 record are synchronized bit to bit, with the standard vertical bit planes, parallel to Y axis. As in the case of type 3, for any level IZ, only the NWPX words starting from word IPWORD, of each frame are to be considered, where IPWORD for the first frame is given as = (IZ - NZLOW) * NWPX + 1. This corresponds to row (IZ - NZLOW + 1) for the frame. These NWPX words have NPX bits (NPX = NWPX * 36), and the successive bits are to be operated with the corresponding bits in the bit plane IZ, in successive rows (see Figure 6.3) between rows NYO and (NYO + NPX - 1). To take a specific instance, the first bit of word IPWORD of frame 1 goes with the bit at the crossing of column NXLOW and row NYO (bit N); the second bit into the crossing of column NXLOW with row NYO + 1, and so on, till the NPX bits are exhausted.

The next frame is synchronized with column NXLOW+1, and hence the bits go into that column in the same way as before. The new IPWORD with respect to MATRIX is computed as

IPWORD = IPWORD, if NPART = 1 (same frame repeats from NXLOW to NXHIGH)

IPWORD = IPWORD + NWORDS, if NPART > 1

This whole procedure is repeated for all IX, from NXLOW to NXHIGH.

Since the operations are at bit level rather than word level, a part of the procedure has been written in assembly language, to improve efficiency and speed. Given IPWORD (starting word number in the frame), NPX (No. of bits to be processed from the frame, starting from IPWORD), the word and bit number of N in Natrix, and NWX (the plant length in words), this routine processes the successive bits in the frame into the said bit number in successive rows of Natrix.

6.5 Type 4

The extraction is more complex for type 4 records. It is done in two stages. For each level IZ, (i) the horizontal section HHHH (Figure 6.4) is extracted first; this is inclined by θ_x to the bit arrangements in the normal bit plane. (ii) the section HHHH is superimposed over the standard bit plane.

The width of plane HHHH is the same as the frames, and is a multiple of 36 bits. Hence, the extraction of HHHH is done by oring together appropriate rows of each frame into the corresponding rows of HHHH, which can be done fast.

In the second phase, for each on bit in HHHH, the corresponding bits affected in the normal bit plane are to be updated [ORed, ANDed or REMOVEd from].

The values XOH and YOH obtained in Phase I give the origin of the frame HHHH. Initially, for each IZ, a frame of suitable size is initialized to zeroes for HHHH.

(i) Extraction: For the IZ being considered, the limits of frames to be considered (See Figure 6.5) are first determined. Out of the frames 1 to N that actually complete the current data section, only frames NLOW to NHIGH are available in the current record. Within this range, the frame numbers LMIN and LMAX, which actually affect layer IZ, are determined. Even if a frame does not actually cut IZ, but its range of influence does, that frame is included for consideration (example, IZ_2, in Figure 6.5).

For each such frame between LMIN and LMAX, the range of rows, IYMIN, IYMAX, that actually contribute to HHHH, are determined (See Figure 6.6).

Each such row J, between IYMIN and IYMAX, is next ored into all those rows (two adjacent rows, at most) in HHHH, whose ranges of influence fall within the range of this row J (See Figure 6.7).

This procedure, when complete, leaves the extracted horizontal pattern in HHHH.

Note 1: Till this stage, there is no fundamental difference needed in considering shapes to be ORed, or ANDed or Removed from; whatever the function to be performed with this object (or primitive) comes in only in the next stage of superimposition. What has been obtained in HHHH now is only a horizontal section of the primitive/object.

However, there is an extra consideration for cases of removing. Due to the noise generated in the above process, the section obtained in HHHH would be larger than the actual object, and if this is removed from the existing patterns (in the next stage), we would be departing from the conventional safer side approach.

Nor is it possible to effectively modify the above procedure, to suit the removing case also, because of the nature of the procedure itself. An alternate procedure is needed, in which, for any row M of HHHH, the surrounding four rows KKKK, obtained from the two engulfing frames (see Figure 6.8) are ANDed together first, and the result is ORed to row M. This ensures that only if all the four surrounding bits are on, the bit in HHHH is turned on, which is in accordance with our conservative approach (for removing case).

Note 2: When $\theta = 90°$, due to the synchronization forced on the rows of the frames with the levels IZ, IYMIN and IYMAX will be equal. This obviously will reduce noise a bit.

(ii) Superimposition: In this stage, there is no scope for utilising the word handling capabilities of the computers and processing has to take place bit by bit. Hence part of the procedure is in machine language.

Since the origin (XOH, YOH) of the frame HHHH is known, the exact position in plan occupied by each bit P of HHHH can be determined, say (x,y) in Figure 6.9. The square of influence of this bit would fall over four bits QQQQ in the standard bit plane IZ, due to lack of synchronisation. (It should be remembered that the plane of frame HHHH is synchronized with the standard bit plane IZ). Hence the bit P is operated over all the four Qs (OR, AND, or Removed from as the case may be.). The bits QQQQ can be easily computed, as

Q_1 = NBIT (x - pitch/2), NBIT (y - pitch/2)

Q_2 = NBIT (x + pitch/2), NBIT (y + pitch/2)

Q_3 = NBIT (x + pitch/2), NBIT (y + pitch/2)

Q_4 = NBIT (x + pitch/2), NBIT (y - pitch/2)

It is obvious that considerable noise is generated in the process, but it cannot be helped. The noise is however only on the safer side, and does not drastically affect the performance. It is safe enough even for removing cases, thanks to the very stringent extraction procedure, and the consequent negative noise; part of this noise is offset in the superimposition phase.

6.6 Relative Speed

An estimation of the volume of computations for each of the above four types would reveal a drastic increase, as we move up the type numbers. Given the same record (containing the same number of frames and same pattern)

for each of the types, the relative computation times for Phase II would be of the following order (calculated approximately from the actual routines):

Type 1	1.00
Type 2	1.25
Type 3	4.50
Type 4	10.00 to 25.00, depending upon θ, θ_x, density of occupation, etc.

Thus, the preference shown to type 1 (and 2), and the degree of conversion to type 4 in Phase I (segmentation), is completely justified. When an object occupies a type 3 or type 4 position, attempt is made to cut it into as many segments as possible, so that the maximum part of the object is classified into the lowest type numbers.

Relative Noise: The noise generated in types 1,2 and 3 are pinned down at the minimal level due to the synchronized bit handling. This minimum noise cannot be further reduced in any way, as it comes in due to the nature of internal representation adopted, and the safe sided approach.

In type 4, the noise is rather high. To give a practical instance, for one particularly bad combination of θ, θ_x and other factors, the object dimensions (in this case, a pipe) got extended by upto three bits, along the edge. However, being only on the safer side, there are no serious problems. If there is a reliable procedure, detecting and suppressing false conflicts (due to noise) from the generated conflicts, this problem can be completely ignored.

6.7 Columns, Beams and Floors

As mentioned earlier, the Phase I and Phase II are merged into one for columns, beams and floors. This is so, because (i) these objects cover the whole plant, and we would do better to take the frame size as the plant size itself, and (ii) the most convenient plane for considering these objects is the horizontal plane. These two are the vital characteristics of Phase II output records.

A set of columns, beams and floors, that lie between the same top and bottom levels, would constitute one record in the Phase II output. These may actually run through a few standard horizontal bit planes, from IZLOW to IZHIGH, but their section along any of these bit planes is just the same, and it is enough that just one such bit pattern section be constructed and stored. The limits of layers IZLOW, IZHIGH are stored in a table INDEX. For each record output, two entries are made in INDEX, = IZLOW and IZHIGH. The total number of records is maintained as a parameter NTOT. INDIM is set to 1, to indicate that INDEX is a two column array.

The limits of affected bit planes for two or more records on the tape may overlap. The Phase III procedure has been designed to take this into consideration. The records are written exactly in the sequential order of input data. All those columns (or beams or floors) that

appear together and have the same IZLOW and IZHIGH are accumulated into one group, and the accumulated bit pattern is written out on tape, when the IZLOW and IZHIGH for the new card are different from those of the old card(s). NTOT is then incremented, and one more row in index is filled with the layer limits of the written out record. A new buffer of 5000 words is then initialized to zeroes for the bit pattern of the next group.

Pattern Construction: In the case of columns, the centre, and the effective web and flange of the columns are known as well as the orientation of the web. Correspondingly, a rectangle with proper orientation and dimensions is inserted with the mentioned point as centre. Note that the 'I' sections are treated as full rectangles, of sides equal to the web and flange of the section.

For beams, the input defines the axis and span of the beams, and the dimensions of the 'I' section. A rectangle is inserted for every beam, with the axis the same as the beam axis, and width = flange of the section or width of the flange plate, whichever is larger.

Each subsection in a section defining floors, defines a floor with a given top of surface and thickness. The extent of the floor can be composed by adding or removing suitable rectangles [oriented parallel to the X, Y axes];

the corresponding bit patterns are added or removed. Each portion being added or removed can be identified. This scheme makes possible a logical approach to defining floors; e.g. we can add the whole plant, and remove rectangles from it for stairs, generator, etc.

For all these three categories of objects, only addition and removal of bit patterns, corresponding to rectangles oriented normal to X,Y axes are required. These are done by a fairly simple routine, that inserts the rectangles given the X and Y limits.

CHAPTER VII

PHASE III EXECUTION - UPDATING STRATEGY

7.1 Outline

When Phase II has been completed, the intermediate output records, consisting of the patterns along standard horizontal bit planes (for the current data section) would be available on AUX1. In Phase III, these records are superimposed (added or removed) over the existing bit patterns along the corresponding bit planes and conflict patterns are composed sequentially, in AUX2, . . .

The table INDEX, as mentioned in the last chapter, gives the layer numbers corresponding to each of the NTOT records on AUX2 (Phase II output). If INDIM = 0, there is one entry per record, and the record affects only that layer. If INDIM = 1, there are two entries per record, giving the lower and upper limits of the layers affected by this record. The parameter INDIM helps generalize the Phase III procedure inspite of the variations in Phases I and II.

The Phase III updating procedures have been devised with a few important considerations in mind, to maximise flexibility (error recovery etc) and speed. They are as below:

(i) Tape operations are minimised. To cut down non data transmitting operations on tape (as search, rewind, etc), the current positions of the various tapes are completely utilized in sequencing the operations. The algorithms are further designed to allow overlap (if available).

(ii) In accordance with the decision that overwriting (main tape) records should be delayed to the maximum possible, the results of the updating (superimposition) are not stored back on the same file; they are stored on a different file. This provides, besides other facilities, a capacity to restart execution from the current data section. The most outdated file is selected for storing the resulting updated records.

(iii) In view of the fixed data structure, and its constraints, updating has to proceed, essentially, file by file, rather than record by record. All the records in a file are updated sequentially, from the first record, rather than in any other order. A file is updated only if at least one of its records (i.e. the corresponding plant level) is affected by the current data section and this minimises the number of files to be updated. (Incidentally, this consideration dictates that the number of records per file is kept minimum).

(iv) The provision of two Main tapes, whatever be the plant size, rather than one, is partly motivated by the following reasoning: The search and positioning operations on the tapes as well as the I/O time can be reduced to the absolute minimum, by (a) locating the new file (for storing

the updated records on a different tape than the old (to be updated) file and, (b) updating the records in the file in the rigid physical sequence, so that once the two files (on the two tapes) have been initially located, no further repositioning is ever required. Also, the two tape units may be put on different channels, and reading in, writing out and updating (computation) can proceed in overlapped mode.

(v) For one data section, there should be only one (minimum) updating operation. No intermediate outputs are ever stored in the main tapes, and only the final updated file is stored on the main tape. This step ensures that the absolute minimum of the main tape files are overwritten, which in turn maximises the storage redundancy and improves performance.

The above steps individually and collectively improve the Phase III performance considerably. It may however be noted that they are only extensions of the basic guidelines mentioned in Chapter II.

The next sections deal with the algorithm in detail, and the significances and the implementation of the above would become more obvious.

7.2 Through Columns

The first file (one record) of tape MAIN2 - numbered as file 0, record 0 - contains a horizontal section of all those columns that run from base to ceiling. Whenever such a culumn(s), termed a 'through column', is added, this record has to be updated.

Maintaining this record saves a lot of effort, at least in the initial stages; when at any level, no object has been added yet this record (containing the through columns) can itself be taken to give the section at the level. By this, (i) duplication of the same record many times (for each level) is avoided, and (ii) whenever a fresh 'through column' is added, updating just this record takes care of all those layers, where no object has been added yet. Thus, considerable I/O is avoided. The saving is particularly high in the initial stages, when very little (or nil) of the objects have been added, and hence, in most of the levels, only the through columns exist. However, this saving holds only for the case when all the columns are completely inserted and checked out at the very beginning, before any other object is inserted. It pays further to insert first all the through columns before the other columns. Whenever possible, it is advisable to convert columns with varying sections to single through columns, either (i) by a safe approximation, or (ii) by extracting a common through column - obviously, the section with minimum dimensions. The larger portions of the same column can later be inserted again (without interference check). This saves some effort.

A Phase II output record is classified as a through column(s), if

(i) INDIM = 1, indicating that the record affects a range of layers, and

(ii) the record affects the whole plant, i.e., from the base layer to the ceiling layer.

All such records out of the NTOT records, that are classified as 'through columns', are first extracted and 'ore'd together. The records are determined by a sequential scan of the table INDEX. When such a record is found, the tape AUX1 is positioned, and the record read into a buffer. When the reading in is complete, it is ored with a scratch buffer, which contains, in the end, the 'or'ed image of all the 'through columns'. For the first such record, the reading in is directed into the scratch buffer itself, to save time.

In the over lapped mode, the selection of the next record to be read in (i.e., further scanning of INDEX), and the ORing of the record already read in, with the scratch buffer, go in parallel with another read in. This needs a systematic switching of the reading in buffer.

Before these operations, MAIN2 is rewound (only if there is atleast one such record). When all the through columns are ready, the existing columns are read into another buffer. The two buffers are superimposed, and the result is stored back on the same file 0 in MAIN2.

7.3 Updating

As mentioned earlier, two parameters, MIN2 and MAX2, which give the lower and upper limits of the layers affected by the NTOT records are computed in Phase II. Updating of main records is carried out only within this range of layers. Below is given an outline of the updating procedure.

(i) Updating is to be done file by file, and each file contains N records, corresponding to a group of contiguous layers in the plant (Chapter III). Hence from MIN2 and MAX2, the limits of group numbers affected (=I1,I2) are determined as

$$I1 = (MIN2 - 1)/N + 1$$

$$I2 = (MAX2 - 1)/N + 1$$

Updating starts from group I1 and proceeds upto group I2.

(ii) For a group of layers, I (ranging from I1, I2), the limits of layer numbers that form this group, are determined first, as

$$N3 = (I-1) * N+1 \text{ and } N4 = N3 + N - 1$$

It is verified next, if any of the NTOT records affect any of the layers of the plant, between N3 and N4. If none of them affects, updating is not done for this group of layers (as for group I in Figure 7.1).

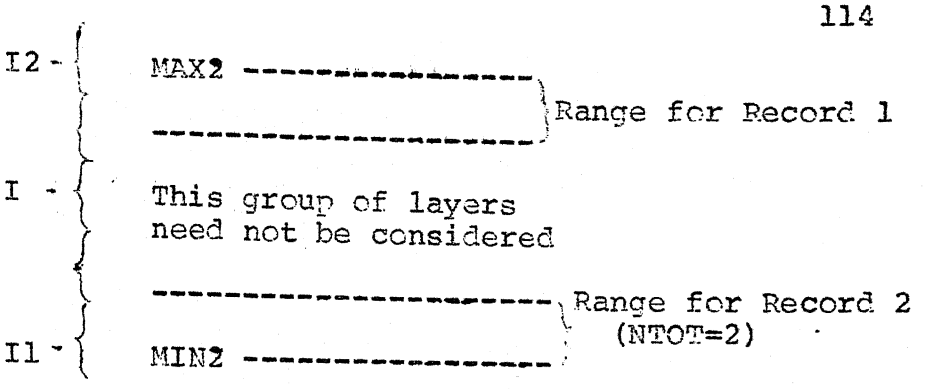


Figure 7.1

(iii) From the file table, the tape (MAIN1 or MAIN2) and the file number of the latest version of the group of layers is obtained, and the particular tape (say MAIN) is positioned at the appropriate point (start of Record 0 of file MFILE).

(iv) Next, the tape and file in which the current group will be stored after updating, are determined. To maintain error recoverability, the updated records are not stored back in the same file. To permit overlap, and avoid too many tape repositioning, the file for the result of the update is located on a different tape (termed MAINA) than the tape MAIN.

Selecting the new file: The most outdated file is chosen, to store the update output, on MAINA to ensure that overwriting the records is delayed to the maximum. The record 0 of the files, containing the stage number NSTG of that file, comes to use here, in that the particular file with lowest stage number can be straightaway chosen.

The actual procedure adopted saves much time, by avoiding a scan of the whole tape, without any addition to the in core tables either. The following parameters are used in the process:

MFLGA = the next file number to be considered on MAINA

MTOTA = total number of files in MAINA (known to the program)

ISCANA = total number of scans to date on MAINA.

MFLGA is a pointer, which remembers the last file number allotted (plus 1) to any group of levels, on MAINA and straightaway gives the next file number to be considered for allotment. At the very start, it is set to 1 (to indicate that file 1 is to be considered for allotment). Evidently, the allotment of files takes place in the physical sequence of the files.

The file table is searched, to determine if file MFLGA of tape MAINA contains the latest configuration for any group of layers. If it does, MFLGA is incremented, and the search continued. If the file is not an entry in the file table, it is selected for storing the updated records for the current group of layers. In case MFLGA exceeds MTOTA, it is reset to 1, ISCANA is incremented (to signify that a rescanning of MAINA has started), and the table search is continued, till a free file is obtained.

The way of selecting a free file obviously will ensure that the most outdated file is chosen. Further, physical tape scanning is avoided, and the amount of repositioning required on MAINA will be minimal (as files are allotted sequentially, and not at random).

(iv) The record 0 of the old file (on MAIN) and the allotted new file (on MAINA) are verified, to make sure there are no mistakes. The record 0 of MAINA is then overwritten by the new status words, as below:

NPRVS = MAIN * 1000 + MFILE (previous file No.)

NSTG = current stage number

MAIN is positioned at record 1, and MAINA is ready to write record 1, by virtue of the above write operation. No more positioning operations will be required on these two tapes, for the current group of layers. The stage number of the file being written over is remembered, as it gives the idea of the maximum possible GOBACK.

In the case that MAIN = MAIN2 and MFILE = 0 (when only through columns are present in the group of layers), record 0 is not verified as for other cases.

The file table entry for group I is next changed to MAIN * 1000 + MFLGA, the new file.

(v) For each group of layers, I, updating is done layer by layer, from N3 to N4. This is also the order in which the records are available in MAIN, and in which they are written out after updating on MAINA.

For each layer, each of the NTOT records (Phase II output) are considered. The table INDEX is scanned sequent ially to determine if a particular record affects the layer or not. If not, that record is not considered. Normally, Phase II output consists of one record per layer in the sequential increasing order, and repositioning of AUX1 is not required. But in the case of beams, columns and floors (INDIM = 1), repositioning may be required, as more than one record may affect a single layer, and they may occur in random.

The first phase II output record, that affects the current layer, is read into a buffer (MATRIX). All the subsequent records (affecting the same layer) are read into a different buffer (NATRIX), and are immediately ORed together w ith MATRIX. (NATRIX for each such subsequent record need not be the same, and the reading in and the ORing of a previous record can be done simulatneously). After all the NTOT records have been considered, MATRIX would contain the accumulated pattern at that layer. This pattern is to be operated over the existing patterns at that layer, as given by the operation code (Add/Remove with/without check).

The existing patterns at the layer, available in MAIN, are read into a buffer (physically different from MATRIX). The two are then merged as required, and the result stored on MAINA. During the process of merging, the conflicting bits are detected and stored sequentially on AUX2, for further processing.

The same procedure is then repeated for the next layer, till all the N layers in the current file (group of layers) have been updated. By virtue of the read operation on MAIN and write on MAINA, for each layer, the tapes would always be ready in position for the new layer. In case MFILE = 0, the tape MAIN (=MAIN2) is rewound for each layer.

It is very often possible that the same pattern (obtained by ORing the same set of Phase II records) has to be merged over a contiguous set of layers. In that case, the reading in of Phase II records, and their ORing together, need not be repeated so many times, if the results are saved after each layer. A look ahead scheme determines the number of layers for which the same pattern gets repeated (i.e. the same Phase II output records are to be ORed together), and for that many layers, this part of the procedure is suspended. It is ensured that the resultant pattern is not destroyed in the middle, so that it can be used repeatedly. This scheme saves a lot of I/O operations.

In the case when MFILE = 0 the old patterns (through columns) are the same, and if the resultant new patterns are also the same for a few adjacent layers, it reduces to writing a particular record (the new pattern) so many times repeatedly on MAINA. (This operation can be performed in full overlap, in the overlapped mode, by specifying with each write command, the number of times the record is to be written out. The overlap monitor would automatically take care of incrementing the record number in the label each time.)

After all the layers in the current group have been updated, the parameter NNEXT in MAIN tape (indicating the next file) is altered to MAINA * 1000 + MFLGA.

7.4 Conflict Messages

The conflict patterns are sequentially written out on AUX2, layer by layer. Only in cases where checking has been requested, and when there is a conflict, these messages are generated. As merging proceeds word by word in the buffers, the messages are generated in the same order and the patterns constructed. No conflict messages are printed out during Phase III. Phase IV and further execution proceed only in case checking for conflicts has been requested. The generation of the conflict bit patterns are explained in a later chapter.

7.5 Post Update Procedures

It was mentioned in Chapter III on the tape structures, that each main file consists of an extra trailer record (of 5000 words) at the end. This record contains the information required to identify the conflicting objects, at a later phase of execution. Essentially, it is a list of all the objects that occupy any space in that groups of layers. Every time the file is updated, by the addition of new objects in this group of layers, the list of objects is also updated. The actual entries made in this list and the way of updating the list, though they form part of Phase III, are explained in Chapter VIII.

Besides this list, there is also one word in this record, that gives the file number of the next updated version of this file (if the file is not its most recent version) - termed the forward link, NNEXT. After the actual updating of the layers is completed (as explained so far), the following are done:

(i) The trailer record of the old file is read in (from tape MAIN). MAIN is backspaced. The forward link (NNEXT) is computed as:

NNEXT = MAINA * 1000 + MFLGA.

The trailer record is written back on MAIN, with this altered value of NNEXT.

(ii) The list of objects is updated.

(iii) NNEXT is made 0 (to indicate that the file on MAINA is the most recent version), and the updated trailer record is written in MAINA.

CHAPTER VIII

PROCESSING CONFLICT MESSAGES

8.1 Outline

The conflict patterns generated as Phase III output are further processed in Phase IV onwards, for suitable interpretations. Execution of Phase IV onwards is carried out only for those cases where checking for conflicts has been requested. Else execution stops for the section with Phase III.

As mentioned in the last chapter, the Phase III output on AUX2 consist of the conflict patterns along those layers where conflicts exist, alongwith the following parameters - (i) layer number, (ii) minimum word number of buffer, that is affected, (iii) Maximum word number affected. There is one record on AUX2 for each layer that has conflicts. Since the layers were updated sequentially in Phase III, the output on AUX2 are also in the ascending order of the layers, so that the missing layers can be immediately obtained without extensive scanning of AUX2. The total number of records on AUX2 is remembered in core.

The following procedures form part of Phase IV:

(i) Suppression of conflicts at requested points, to the required extent [SUPRES card].

(ii) Storing of the resultant conflict pattern in main tapes, if requested for.

(iii) Printout of the conflict patters in an easily interpretable form, if requested for.

In Phase V, the conflicting objects (if any) are identified, if the user has asked for the same, and in Phase VI, the pipe path is adjusted to circumvent conflicts.

8.2 Conflict Suppression

It was mentioned in Chapter IV that the objects within a data section are not checked for conflicts against each other. This effectively prevents the generation of conflict messages around every pipe joint and there will be so many of them in a system. But the piping system itself (represented by the section) has to be connected to outside elements as boilers, pumps etc., and these objects cannot also be included within the same data section. Hence, a scheme for suppressing such conflicts becomes essential. The SUPRES card helps in this; the point where suppression is desired (say joint between a boiler and a pipe) and the extent of suppression along X, Y and Z directions around this point, are specificed in each SUPRES card.

The suppression mechanism is based on a table is core. The table contains 6 entries for every point of suppression, which give the X, Y and Z bit limits, between which the conflicts are to be suppressed. (These bit limits can be easily obtained by transformation from the real space limits). The maximum number of suppression points depends on the table length, and can be varied during system assembly.

Each record on AUX2, containing the conflict pattern along a plane, is checked against this suppression table. All those areas specified in the table are made zeroes in the frame, and further processing is done with the resulting frame. If however the resulting frame is all zeroes, the particular layer has no conflicts.

8.3 Storing the Conflict Patterns

In most cases, the identity of the conflicting objects will be obvious even without a printout of the patterns. However, to be on the safer side, the conflict patterns may be stored in the main tapes (at the request of the user) for retrieval in the next run, in case the objects could not be guessed out. This step may save a lot of time (and pages) otherwise wasted in generating printouts. The position in Main tapes where the pattern is stored is printed out for the user. The availability of the patterns after the immediate next run is not however guaranteed - as it is not felt necessary.

The file on the Main tapes for dumping these patterns is selected the same way as in updating (the most outdated file is chosen). More than one file is taken up for the purpose, if necessary (when there are many frames of conflict patterns). The actual patterns stored are the remainder ones after the suppression phase is over.

8.4 Print Out Generation (of Conflicts)

The conflicts patterns at each bit plane (where conflicts exist) are printed out only when asked for. The conflict patters would be available, at this stage, in core (say in buffer MATRIX). From the file table, the latest updated pattern for the same layer is read into NATRIX, from the main tapes. The printout is generated in very similar fashion to the normal printout generation and the same routines are used with slight changes in parameters (see Chapter IX). The reference axes are printed on the sides as before. The off bits in NATRIX are printed blank, the on bits of NATRIX are substituted by 'X's, and on this pattern, those bits which are on in MATRIX are changed to 'C's. What results then is a three tone printout.

The form of printout can be further improved but at the cost of execution time. To avoid too much time being wasted in generating printouts, the improvements are not attempted.

Only that portion of the frame containing the conflicts is actually printed out (with a reasonable margin on all sides). The extent of printout is easily computed from the minimum and maximum word numbers having conflicts (available alongwith the conflict patterns).

8.5 Conflicting Objects Identification

The identification of the conflicting objects from the patterns may be attempted in two ways:

(i) By attempting a reconstruction of relevant objects, and selecting the ones that contribute to the conflict patterns.

(ii) By pattern recognition methods, the conflict patterns may be decoded.

The latter method may not be reliable, due to the excessive digitization, and the similarity in bit representation of most of the objects. Pinpointing the exact object (and its data card) may not be possible. Hence the first method is chosen.

The trailer record in each file is essentially meant to assist in the identification of the objects. This record consists, in a convenient form, a list of all the objects in that group of layers. The plant space occupied by each group of layers is split into standard partitions, which are assigned numbers. The entries in the trailer record are in the form of strings, as: - stage number, <list of partition numbers>. Such a string indicates that objects

within that stage number affect the listed partitions. The -ve sign for stage number is a flag, that indicates that the number is a stage number. Each time an object is added to this group of layers, the record is updated suitably. Entries are however not made for through columns, which exist throughout the plant. In the case of beams, the list of partitions does not exist; only the stage number is entered. The composing of the entries for each section is not very tough. Incidentally, the list need not be accurate; it is enough that it be on the safer side.

As a first step in identification, the conflict pattern is anded with the through columns. The result, if non-zero, shows up the conflicts with columns, and the individual columns can be easily located by a scan through all the columns.

Next, the partition numbers of the conflicts are determined. From the trailer record of the old file, a list of possible stages (which affect these partitions) is drawn. Each of these stages is considered one by one, and the patterns generated by that stage, on the particular layer and partition numbers is determined. This is anded with the conflict patterns, and the result gives the contribution of that stage to the conflict. This is repeated for each of the stages in the list.

Each time a part of the conflict has been accounted for by an object, the corresponding pattern is removed from the conflict patterns, and analysis proceeds with what remains.

The reconsideration of so many stages will not be very slow, as only one frame (just a part of it) has to be constructed. The data cards for these old sections are taken from tape NPRO.

8.6 Adjusting the Pipe Path

The actual algorithms for adjusting the pipe path have not been developed. But a few observations can be made on them at this stage. First, an interactive mode, where the computer takes on line guidance from the engineer, would be more plausible. It might simplify the algorithm, and would also utilise the engineer's experience.

The adjusting algorithms should be based on the information collected on the previous phases on the flexibility of adjustments in individual layers, rather than go about reconsidering each layer. This would reduce a lot of I/O.

And lastly, the algorithm has to be an effective and fast one, rather than a trial and error procedure of considering all possible adjustments.

CHAPTER IX

BASIC PROCEDURES

9.1 Lines

Most of the pattern constructing algorithms in the package ultimately boil down to inserting lines (strings of bits) at appropriate points in the frames. As such, the routine that inserts (or removes) lines has to be extremely optimised in execution time, besides catering to the normal requirements (as checking for conflicts, etc.) in operating with lines. For efficiency, assembly language has been used.

In calling the routine, the following parameters are to be specified -

(i) Starting machine address of the frame (say MATRIX),

(ii) Bit limits of the line (These bit numbers are computed with respect to the frame origin. Figure 9.1 shows how they are determined).

(iii) Function to be performed, as adding (ORing), ANDing or removing from, with or without check for conflicts.

The line may actually run through a few machine words. For each such word, the pattern - a string of '1's - is first constructed. The pattern will be right adjusted for first word, complete for intermediate words, and left adjusted for the last word (at times, there may be only one word affected). Next, depending upon the operation to be performed, one of the following procedures is followed:

(i) OR, no check - OR this pattern with the appropriate word in MATRIX.

(ii) AND - AND this pattern with the appropriate word in MATRIX.

(iii) Remove from - Complement the pattern and AND the result with the appropriate word in MATRIX.

(iv) OR, check - (a) get a copy of the pattern. AND the copy with the word in MATRIX. The result gives the conflict pattern (no conflict if all bits are zeroes).

(b) OR the pattern with the word in MATRIX.

(v) Remove, check - (a) get a copy of the pattern. Complement this copy, and OR the result with the word in MATRIX. Complement. The resulting pattern, if non-zero, gives the conflicts (i.e. bits being removed were not already on).

(b) Complement the pattern, and AND the result with the word in MATRIX.

The conflict patterns generated are available in standard locations, and are used by the calling routines.

9.2 Inclined Rectangle

Input to this routine are -

(i) The address of the origin, and the frame dimension NPX (bits) of the frame (say MATRIX),

(ii) Points A and B - the ends of the axis of the rectangle, and

(iii) Width W (see Figure 9.2)

(a) The lowest and highest of the row numbers affected by the rectangle are first calculated (=NPYMIN, NPYMAX), as also the position of points 1 and 6.

(b) Figure 1536 is constructed, by inserting lines LL in each affected row. Since the width a is constant, this pattern can be constructed very fast.

(c) Areas 152 and 643 are removed (by removing lines in those rows).

What remains is rectangle 1 2 3 4. Special care has to be taken when the points 5 or 6 fall outside the frame and when slope AB is very low. Also, while removing the two triangles, care is taken not to remove too much of the lines (safe sided approach), along edges 12 and 34.

Only creating such rectangles is possible. When other functions as adding or removing are required, the rectangle can be constructed in a frame, and the frame can be merged with the existing patterns.

9.3 Merging

Given two frames of patterns, the routine 'Merges' one frame into the other - the variations possible are, OR, AND, REMOVE, with or without check. The arguments are -

(i) Merging frame origin (MATRIX),

(ii) The sink frame origin (NATRIX)

(iii) The extent of merging (Number of machine words to be merged); and

(iv) Function to be performed.

In the end, MATRIX is left undisturbed. The origins mentioned need not necessarily be exact frame origins - they may be shifted [e.g. MATRIX (141)].

For each word in MATRIX, to be merged into a corresponding word in NATRIX, one of the following actions is performed:

(i) OR - OR the MATRIX word into the NATRIX word.

(ii) AND - AND the MATRIX word into the NATRIX word.

(iii) Remove - Complement the MATRIX word, AND it into the NATRIX word.

(iv) OR, check- (a) AND the MATRIX word and the NATRIX word the result (if non-zero) gives the conflict pattern at that word.

(b) OR the MATRIX word into the NATRIX word.

(v) Remove, Check - (a) Complement the MATRIX word, OR it with NATRIX word, and complement the result. What remains, if non-zero, gives the conflicts.

(b) Complement the MATRIX word, and AND it into the NATRIX word. The conflict patterns generated are left in standard locations, for each word.

9.4 Printout Generation

Given a frame MATRIX and its dimensions, a list of defined axes, and the limits of print out required, this routine generates the printout.

The printout is actually split into vertical strips, each strip 120 bits wide, and the strips are printed out one after the other, left to right. Internally, the print out is generated row by row (across all these strips). The first strip is printed out immediately and the remaining strips are stored on tape for subsequent printing.

Before the actual rows start, there is a row of defined X axes; the names of the axes are plugged into this row at suitable points. While each row is generated, if any defined Y axis falls within the range of that row, its name is printed at the start of the row. Else the place is left vacant.

The generation of each row is actually a blowing up of the binary pattern to a BCD pattern each bit is corresponded to a character in the printout. A blank is generated corresponding to a '0' in the pattern, and an 'X' for a '1'.

A slight variation of the procedure is adopted for the multi shade printout for conflicts. There is an extra frame in this case (NATRIX) that has the conflict patterns. The frame MATRIX has the resultant merged patterns. The rows are constructed the same way as for the above case, but before printout, a scan of NATRIX is made. If any

bit in NATRIX is on, the corresponding character in the printout is changed to a 'C'.

9.5 Tape Operations

A base level of input/output, namely IOOP (Input Output Operations, in IBM 7044) is used for tape operations, in view of the following facts:

(a) It permits the specification of I/O buffer area

(b) It permits non-standard recovery procedures,

(c) It permits direct control over priorities of operations, and makes over lapping possible. Also, maintaining a labelling scheme becomes much easier.

For any operation on tape (both I/O and positioning), the following parameters should be specified in the calling sequence:

(i) tape number, file number and record number [TAPE, FILE, RECD]

(ii) Starting address (if I/O operation) [MATRIX]

(iii) Word count - +ve for write, -ve for Read, 0 for positioning [COUNT].

(iv) Whether to ascertain the current tape position before I/O [CHECK=1] or not [CHECK=0].

The routine checks the current tape position and suitably repositions the tape, if an I/O has been requested with CHECK=1 or if COUNT=0. For an I/O with CHECK=0, the prepositioning is not done. It is taken as an immediate read/write operation. Only if the first trial results in error, CHECK is automatically

made = 1, and further attempts are made with proper repositioning.

The first three parameters are collapsed into a single word, LABEL (Chapter III). The first word of each record on tape is the label word, that defines the current position of the tape. Comparing LABEL with the label word of the current record gives an idea of the direction and amount of repositioning.

The flowchart attached gives the actual procedures adopted. When any tape operation results in error, the system gives an error return to a specified location, and the status of the operation are left in a standard location, from which the cause of the error, and hence the actions to be taken, can be determined. The routine tries each operation ten times before giving up,though normally two attempts are always enough (unless the record is irretrievable). In the first five attempts, the tape is backspaced each time (for tape cleaning) by 4 records, and repositioned. For the next five attempts, the tape is forward spaced by 4 records and repositioned. In the case of the operations being performed on the tail end of the tape, backspacing ensures that the tape does not go to the unlabelled scratch area, as control over its position will then be lost. The calling routine should also take care that the tape is not finally left in the slack space at the end of each file.

In the chart, DUMMY is a scratch area in core, where the label part of the current record in tape is read in for comparision with LABEL.

Often, the tapes have to be backspaced immediately after a read/write operation. A separate entry point (BSR in figure) is provided for this purpose. If the first attempt at backspacing results in an error, a whole sequence of repositioning operations takes place.

CHAPTER X

THE OVERLAY MONITOR

10.1 System Generation

Before the package can be used, the tape AUX1 is to be prepared with the core images, by the standard system generation procedure (explained in the supplement). In essense, it performs the following in the sequence mentioned:

(i) The user assigns values to the alterable parameters in the system data area (listed out in the supplement). These data include the tape assignments, table sizes, etc., and they need be altered from the standard settings only when necessary. The system data area forms part of the 'bootstrap' routine (Section 10.2), which includes the input/output routines also, and resides in the common non-overlay area of memory. This routine is compiled next, to obtain its relocatable deck.

(ii) With this bootstrap routine placed first, all the relocatable decks provided are put on tape in the order specified. This can be done by an 'update' run, as shown in the standard deck set up. The resulting tape now contains all the routines of our system in relocatable form.

(iii) The AUX1 building up is done now, in three phases (corresponding to the three coreloads). Let the core loads be A,B and C. In the first phase, the routines for core load A are loaded in after the common routines from the relocatable routine tape. The routine which supervises the AUX1 building also resides in the common area. AUX1 is rewound, and a dummy file is written out (file 0), of enough length for the snapshot dump. Next, the common program area is written out in record 0, of file 1, after which the core load A (after the common area) is written out in record 1. The common non overlay area runs to about 4000 words (after the system and data areas), and each of the overlay core loads take about 6000 words, after the non-overlay area (see Figure 10.1).

In Phase II of system generation, the routines of core load B are loaded into core from the relocatable routine tape, after the common routines. The core load B part is then written out as a single record, and numbered as record 2 of file 1.

In phase III, the above procedure is repeated for core load C. To be on the safer side, the records 0 to 3 of file 1 are read again and repeated after record. 3. In case some of the first copies of these records are spoilt, the program automatically reads in these second copies, avoiding the need for a rebuilding of AUX1.

After this duplication, a file mark is written to signif end of file 1. A dummy record [file 2, record 0] follows, then another file mark and a dummy record [file 3, record 0]. These operations complete the system generation.

In using the package later, only the bootstrap routine [in binary form] is to be fed, after mounting AUX1 correctly. The program and overlay core loads are read into core by this routine by itself. The relocatable routines tape need not be mounted.

10.2 The Bootstrap Mechanism

Besides the individual overlay core loads, the common routines (including library subroutines) with their linkages have also been assigned space on AUX1, and are read into co at the start of each run by the bootstrap mechanism. This minimises the number of cards to be fed each time (to around 30) and simplifies the deck setup, without adding to the number of tapes. Some time is also saved in loading the common routines each time.

A self overlaying bootstrap mechanism is adopted. The routine that does the reading in resides in the tail end of the bootstrap routine (which includes the program data area of about 16K in it - area III in Figure 10.1). The followin steps help overcome the problems of linkage:

(i) There is no linkage between area II and the other areas (below it).

(ii) All the linkages in area III are to the library routines only (which are loaded by the system in area IV) and are overwritten during the bootstrapping.

(iii) Once the common routines (area IV) have been successfully read in, there are no linkage problems (this is discussed in 10.3).

The area III of core before and after the bootstrapping is essentially the same, but for the linkages to library routines. The routines in area III perform the following functions:

(1) Read in areas III and IV from tape AUX1, file 1, record 0. The actual reading in is performed by the input output routines in area II, and hence control returns to area III only after the reading in has been completed and checked out.

(2) Transfer control to the first location of area IV (the main program).

Once the bootstrapping is complete and control has been transferred to area IV, area III never regains control. The routine in area III is not however completely wipped off, as it will complicate the bootstrapping mechanism (and the linkage establishment).

10.3 Establishing the Linkages

Figure 10.2 shows the linkage scheme in perspective. The features of the system are:

(a) All the library routines are enclosed in the common non overlay area. This is done by loading these routines alongwith the common routines ourself, rather than allow the system to load them at the end. The common routines have the same length for each overlay, and hence these library routines occupy exactly the same locations in core for each overlay. To ensure that the location of the common routines and library routines is invarient for each overlay, they should be loaded rigidly in the same sequence, while composing each overlay core-load (during system generation).

(b) It is ensured that core loads A,B and C are completely d isjoint; i.e., the routines in any one of them are not linked with any routine in the other two coreloads. A linkage between them is still possible, however, through the common routines.

(c) The common routines have to remain in-variant in all respects, when they are loaded in core with overlays A,B or C, i.e., the linkages in the common routines to routines in the core loads A,B and C should not change, whichever core load resides in the overlay area. This is accomplished by avoiding direct linkages between the common routines and the overlay routines. All the linkages are directed through the linkage editors. Each overlay core load has a linkage editor at the top of it.

For each entry point in a core load (say A1 in core load A - See Figure 10.2), there is a routing point in the corresponding linkage editor (SA1). The routine in the non overlay area are linked to A1 through SA1- the CALL A1 statements become CALL SA1. The linkage editor however completely simulates a direct call from the main routines to A1 (as explained in the figure). The return of control from A1 to the main routines takes place directly, and not through the linkage editor. Thus the linkage editor of each core load has a set of routing entry points (like SA1) corresponding to each of the entry points (A1) in that core load.

When the core load A is being assembled in core, the linkages in main to the other two core loads has to be taken care of. Further these linkages should be the same as the ones when the respective core loads would be assembled in core. This is achieved by standardizing the routing points. The routing points SA1, SB1, and SC1, for example, are allotted the same location in all the three linkage editors, are defined in each of them, though SA1 has significance only for core load A, SB1 for core load B and so on. Thus, the links in main to these three routines will be pointing to the same location. However, when routine A1 is to be called core load A is read in and SA1 is called, if B1 is to be called, core load B is read in and SB1 is called, and so on. The reading in of the appropriate core loads is done by the overlay monitor, before control is transferred to location SA1 (= SB1 = SC1).

In a similar fashion, there are more sets of routing points in the linkage editor, as SA2 = SB2 = SC2, SA3 = SB3 = SC3 and so on, depending on the number of entry points in the core loads.

The linking between the routines in the individual core loads with library routines does not pose a problem, as these library routines reside in the non-overlay area, and linkages to them are in-variant. Also, the linkage amongst routines in the same core load need not pass through the linkage editor.

(iv) The overlay monitor : Before passing control to any of the routines in the core loads, the program has to get the appropriate core load read into core (if it is not already resident in core). This is done by the overlay monitor. The main routine calls the overlay monitor and requests the appropriate core load to be read in. The monitor verifies if the particular core load is already in core, and if not, reads in the core load from AUX1. A parameter in core remembers which core load is currently in core. The overlay monitor resides at the tail end of the non-overlay area.

10.4 Selecting the Core Loads

The routines that compose each of the core loads are selected in such a way that the linkages between the core loads are kept to the minimum. These linkages are routed always through the main routines. In a multiphase execution

(as in our case), a logical solution would be to house the routines for the individual phases in separate core loads. This however cannot be followed strictly, as the size of the core loads would become very much unbalanced; the routines for Phase I would occupy perhaps three times the core requirement for any of the other phases. Having the overlay area big enough to accomodate the largest phase would amount to bad utilisation of core. Having the overlay area too small would, on the other hand, increase the number of overlays and slow down the execution.

The solution lies in between the two extremes. After a study of the amount of inter-routine linkages, and the probable frequency of their occurrance (as some of the phases may not be executed for every section), the following composition was decided upon:

Core Load A: Tape initialisation routines, predefinition of standards, printout generation routines (including conflict pattern printout of Phase IV), fatal error diagnosis, and normal termination procedures, and other Phase IV routines [This core load is used the minimum number of times].

Core Load B: Phase I routines for bracings, pipes, objects and object predefinitions.

Core Load C: Phase I routines for floors, beams, and columns and Phase II and Phase III routines.

The routines used for inserting rectangles and lines form part of both core loads B and C.

The routines in the non overlay area are (i) the bootstrap routine, including the input/output procedures and the system data area, (ii) the main routine that supervises the execution, (iii) routines for transformations from real space to bit space and back, (iv) a part of the merging routines, (v) the overlay monitor, and (vi) routines that handle the error conditions (in data or elsewhere).

10.5 Data Transfer Between Core Loads

When some of the results of execution of one core load are required by the next core load, it has to be ensured that these data are in the non overlay area. If this precaution were not taken, the data would be overlaid by the new core load, and would be lost. In this respect, a general consolidation of all the data area in the package (as explained in Chapter III) and locating them in the non overlay area helps remove a lot of confusion.

CHAPTER XI

ERROR RECOVERY

11.1 Maintaining Recoverability

The volume of computation in a typical piping project would be extremely high, running into a couple of hours. It would be a capital loss if all the computations are to go waste, in case of a machine malfunctioning or other errors due to carelessness. Maintaining a capacity to recover from any mishap has been a primary consideration throughout the package design. The recovery speed is also an important consideration, since a recovery procedure that takes as much time as the computation itself is not really worth the bargain.

In order that recovery is always possible, it is ensured that none of the data is held too vital, in the sense, we could always afford to loose it. This is accomplished in some cases by storing the data in duplicate, and in others, by making it easily recomputable any moment. An example of the first is the maintaining of the two almost identical store and snopshot areas, for the basic parameters. An example of the second step is the maintenance of all the data cards to date, in the tape NPRO, so that it is always possible to recompute anything, even without the user's initia-

A further step is to seperate the duplicates (or the successive results) as much as possible. For example, it is simple probability that chances of two tapes getting simultaneously damaged is far more than chances of one getting damaged. Hence (and for other reasons too) the store and snapshot area are located on different tapes. So, also, the successive updated stages of any main tape file.

As mentioned in earlier chapters, it is ensured that the preceding stages of updating of the various layers of the plant are always available on the tapes. This makes it possible that in case the latest version of a layer is lost, it can be instantly recomputed from its previous version, and the data cards. This recomputation would take much less time than a whole recomputation from the start, for that layer. In fact, the recovery procedures are such that even if one full tape (any of the four tapes) gets lost, recovery is still possible.

There is obviously, a price to be paid for these safety measures - in the increased execution time. The more the duplication of storage, the more is the time taken. Hence whenever the duplication effort becomes very extensive compared to the effort that would be required for recomputation, the latter is preferred. Evidently, recovery would be much slower in the case of reconstruction. A balanced approach is therefore required

in deciding which of the procedures is to be adopted for a particular data.

Of the possible error conditions, abnormal termination due to machine malfunction is the most likely. The recovery from such a condition has hence been given maximum consideration. The recovery time depends much on the volume of recomputation required, and this is kept minimum by taking snapshots at every significant moment (as end of every phase for each section). Recomputation is thus restricted to the current phase only. It is ensured that the data required for restarting from the current phase is always available in the tapes.

The taking of a snapshot takes very little time, as the tape AUX1 will always be near load point (the snapshot area). But taking a store would take a much longer time, as it involves rewinding MAIN1, which may be stationed at any point in its full length. Hence a frequent store is not attempted.

It has been observed in practice, that the overhead involved in maintaining the duplication and recomputability is extremely small.

11.2 Standard procedures

Most of the error conditions are detected internally by the program itself, and since all the required information

are available, it performs automatically the appropriate recovery procedures. Only in exceptional cases, when automatic recovery is not possible, manual recovery is requested by the program. As mentioned in the supplement, even for such unusual cases, the manual recovery procedures are very standard and simple.

The possible error conditions, alongwith the recovery procedure (manual/automatic) are discussed in the supplement.

CHAPTER XII

FURTHER IMPROVEMENTS

12.1 A Disk Based System

Converting the package to a disk based system would theoretically improve the performance a good extent, since no time will be wasted in mounting the tapes, and the chances of failure of the disk, or the scartching of a record on disk are much less. But the main factor against having a disk based system is the relatively low memory capacity of a disk unit. A 1301 II disk can at best hold five tape loads of information. Hence, basing the package completely on disk would require that about 40% of the unit be reserved exclusively for a few weeks for this package. This however is normally impractical, and even if it could be arranged, it is not sound to go under the assumption that 40% of a 1301 disk would be always available on demand.

A notable advantage in using the disk for the main records is the complete randomness of access possible. There are none of the problems met with in tapes [Chapter III], and the data structure can be very much simplified. Due to the random access capability, search time can be drastically cut. This would however need basic modifications in the input output routines, and the main data structure.

Even if the main records are not transformed to disk, some saving could still be effected with little extra effort, by assigning tapes MPRO, AUX1 and AUX2 to the disk. IOOP allows disk to be handled in sequential mode as tapes, and no basic change will be necessary. Only, specific areas in the disk should be reserved for the two tapes, and the units suitably switched. These areas should be protected from other users. By this, mounting of two of the four tapes can be avoided, in each run.

12.2 Overlapping

Since the amount of input/output is very large compared to computations, proper overlapping among the I/O, and with computation, would reduce execution time by atleast 50%. All the routines have been devised with an eye on overlapping the operations. As mentioned, a modified cyclic buffering scheme is proposed, using three buffers. A majority of the routines use two buffers simultaneously, a good number use just one and a few of them use all three. To permit maximum overlapping, the sequence of switching the buffers between input-processing-output is slightly modified to suit each routine. The actual steps taken in the algorithms have been explained in this report at the appropriate points.

The structure of each of the three buffers is shown in Figure 12.1. Very often in the algorithms, the same record has to be written more than once, with only the label part (record number) incremented. The repeat factor remembers the

number of times the record is to be written; the input output routine automatically takes care of decrementing repeat count, incrementing label and writing out, each time, till the repeat count becomes zero. The repeat count is also written out alongwith the record, for possible assistance in recovery and updating operations.

The first word of the buffer is the in use indicator. Its interpretations are shown in the figure. The calling sequence to the I/O routines specifies besides the repeat count, the value to which this indicator should be set, after the operation is complete.

The allotment of buffers to the routines takes place on demand. The first buffer which is free is allotted to the routine. At times, if the routine needs more buffers than are available, it will have to wait till an I/O is completed. The buffer allotment is made through a call to the buffer scheduler, which resides in the non-overlay area of the memory.

The IBM 7044 system allows four priority levels, in queueing up the I/O operations. Writing out with repeat count are queued up at the lowest priority (1). Loading a overlay core load is done at the top most priority (4) - the computer essentially waits till the core load has been read in completely. Taking a snapshot also has priority 4. In the few cases, where the completion of a reading in is essential before the computer can proceed further, the

operations are assigned priority level 3. In all the other cases, the lowest order priority is used. When an operation is required on a tape for which there are entries in the queue the latter request is never given a priority higher than the ones before it.

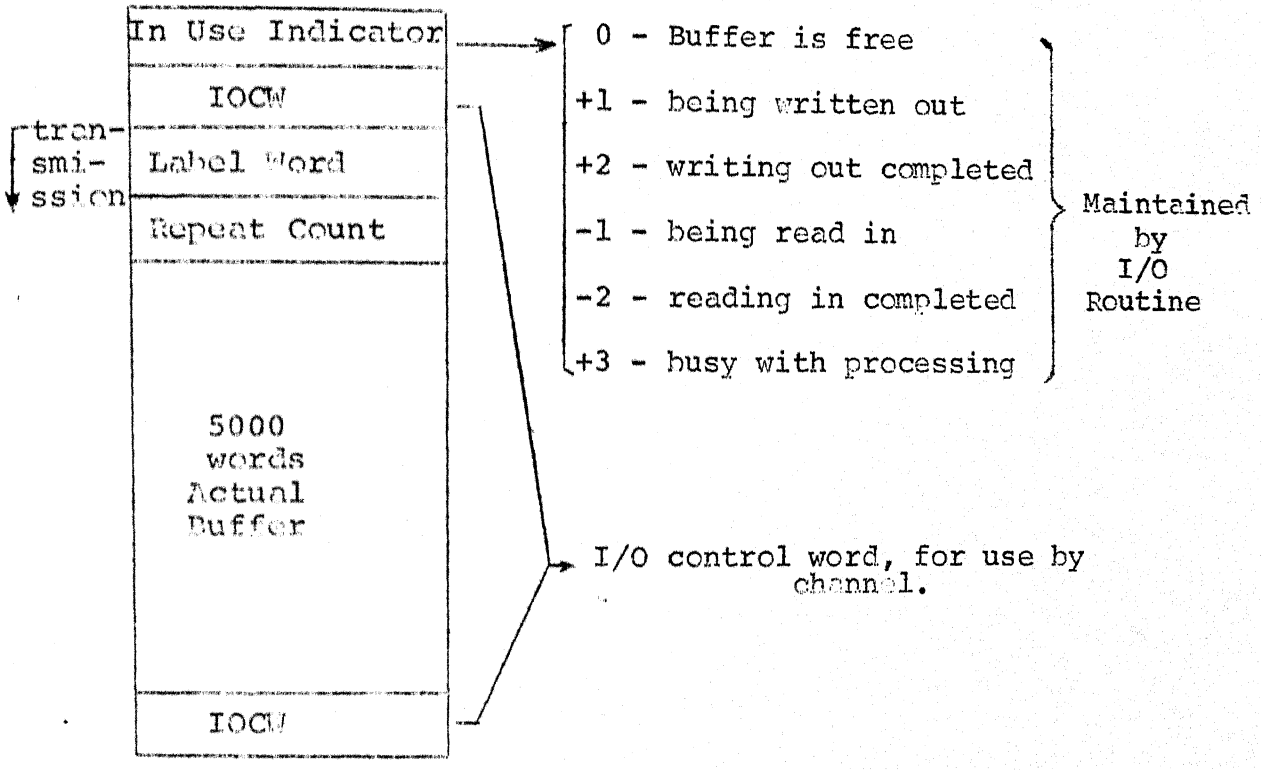


Figure 12.1

12.3 Checking for Minimum Pipe Clearance

There are a few possible ways in which the clearance around pipes can be checked. One such is discussed below.

The pipe section may be treated as two sections; one has the pipe dimensions enlarged to the extent of minimum clearance required, whereas the other has the actual dimensions. The Phase I and Phase II output are duplicated -

the output records for the two sections are generated simultaneously. In Phase III, the conflict check is done with the enlarged pipe section, whereas actual updating is done with the other section.

By this procedure, it is not possible to consider the clearance requirements of pipes already put in. It would hence be necessary that pipes are inserted in the decreasing order of clearance requirements.

The clearance problem can be solved easily even otherwise, by enlarging the pipe dimensions in the input. In some instances however, this may not be a good solution.

12.4 Possible Extensions

There is immense scope for making the package more versatile and powerful. Some of the possibilities are:

(i) Estimation of exact clearance between objects.

(ii) Adjusting the pipe path to avoid conflicts, with the assistance of the engineer.

(iii) Production of piping schedules over an on line/off line plotter, for the erection engineers.

(iv) Extending the package to an interactive mode, over a graphic display.

(v) Assistance in fixing instruments and supports.

(vi) Documentation, inventory maintainance and accounting (limited extent)-

(vii) Automatic pipe layout, given terminal points - under the directions of the engineer.